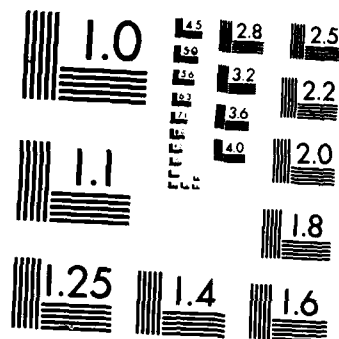


KN05 (KNOWLEDGE-BASED SYSTEM) - THE FINAL REPORT
(1982)(U) MITRE CORP BEDFORD MA R H BROWN ET AL
AUG 86 RADC-TR-86-95 F19628-79-C-0001

(1982)(U) MITRE CORP BEDFORD MA R H BROWN ET AL
AUG 86 RADC-TR-86-95 F19628-79-C-0001

F/G 5/2

44



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

12

RADC-TR-86-95
Final Technical Report
August 1986

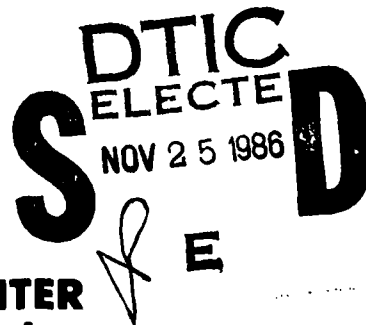


KNOBS — THE FINAL REPORT (1982)

MITRE Corporation

Richard Henry Brown, Jonathan K. Millen and Ethan A. Scarl

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED



ROME AIR DEVELOPMENT CENTER
Air Force Systems Command
Griffiss Air Force Base, NY 13441-5700

86 11 25 031

AD-A174 375

DTIC FILE COPY

This report has been reviewed by the RADC Public Affairs Office (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RADC-TR-86-95 has been reviewed and is approved for publication.

APPROVED:

Northrup Fowler III

NORTHROP FOWLER, III
Project Engineer

APPROVED:

Raymond P. Urtz, Jr.

RAYMOND P. URTZ, JR.
Technical Director
Command and Control Division

FOR THE COMMANDER:

Richard W. Pouliot

RICHARD W. POULIOT
Plans and Programs Division

DESTRUCTION NOTICE - For classified documents, follow the procedure in DOD 5200.22-M, Industrial Security Manual, Section II-19 or DOD 5200.1-R, Information Security Program Regulation, Chapter IX. For unclassified, limited documents, destroy by any method that will prevent disclosure of contents or reconstruction of the document.

If your address has changed or if you wish to be removed from the RADC mailing list, or if the addressee is no longer employed by your organization, please notify RADC (COES) Griffiss AFB NY 13441-5700. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document requires that it be returned.

AD-A174375

REPORT DOCUMENTATION PAGE

1a REPORT SECURITY CLASSIFICATION UNCLASSIFIED		1b. RESTRICTIVE MARKINGS N/A	
2a SECURITY CLASSIFICATION AUTHORITY N/A		3 DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited	
2b DECLASSIFICATION/DOWNGRADING SCHEDULE N/A			
4 PERFORMING ORGANIZATION REPORT NUMBER(S) N/A		5. MONITORING ORGANIZATION REPORT NUMBER(S) RADC-TR-86-95	
6a. NAME OF PERFORMING ORGANIZATION MITRE Corporation	6b OFFICE SYMBOL (if applicable)	7a. NAME OF MONITORING ORGANIZATION Rome Air Development Center (COES)	
6c. ADDRESS (City, State, and ZIP Code) PO Box 208 Bedford MA 01731		7b ADDRESS (City, State, and ZIP Code) Griffiss AFB NY 13441-5700	
8a NAME OF FUNDING/SPONSORING ORGANIZATION Rome Air Development Center	8b OFFICE SYMBOL (if applicable) COES	9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER F19628-79-C-0001	
8c. ADDRESS (City, State, and ZIP Code) Griffiss AFB NY 13441-5700		10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO 62702F	PROJECT NO 5581
		TASK NO 21	WORK UNIT ACCESSION NO 14
11 TITLE (Include Security Classification) KNOBS - The Final Report (1982)			
12 PERSONAL AUTHOR(S) Richard Henry Brown, Jonathan K. Millen, Ethan A. Scarl			
13a TYPE OF REPORT Final	13b. TIME COVERED FROM Oct 79 TO Sep 82	14. DATE OF REPORT (Year, Month, Day) August 1986	15 PAGE COUNT 110
16 SUPPLEMENTARY NOTATION N/A			
17 COSATI CODES FIELD GROUP SUB-GROUP 09 02		18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number) Artificial Intelligence Knowledge-Based Systems Expert Systems Natural Language Understanding Planning mission planning	
19 ABSTRACT (Continue on reverse if necessary and identify by block number) KNOBS (Knowledge-Based System) was a pioneering demonstration of several capabilities desirable in future military mission planning systems. This system was the first to unite a frame representation for data, rule-based inference, English Language understanding using a conceptual-dependency dictionary-based parser for question-answering and control, and a constraint-based approach to planning. Beyond the first AI system for military planning that worked with a realistically sized database, two important contributions of the KNOBS effort were the identification of "stereotypical planning" as an important special case of problem-solving which could be automated, and the notion of a "mixed initiative" where labor was dynamically divided by the user between the user and the machine. This document is a case study of critical decisions that were made during the period 1978 to 1982 in the development of the KNOBS system from the perspective on continuing work in (over)			
20 DISTRIBUTION/AVAILABILITY OF ABSTRACT <input checked="" type="checkbox"/> UNCLASSIFIED/UNLIMITED <input type="checkbox"/> SAME AS RPT <input type="checkbox"/> DTIC USERS		21 ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED	
22a NAME OF RESPONSIBLE INDIVIDUAL Northrup Fowler, III		22b TELEPHONE (Include Area Code) (315) 330-7794	22c OFFICE SYMBOL RADC (COES)

UNCLASSIFIED

Block 19. Abstract (Cont'd)

1986. Some of the decisions led to fundamental new insights (published here for the first time); others were good ideas that just didn't work out as expected. This report is a frank evaluation of both kinds of decisions.

UNCLASSIFIED

Acknowledgements

A number of people contributed to the research effort herein described. The long and patient support of Northrup Fowler, III at Rome Air Development Center made this effort possible. His keen questions and insight greatly increased the quality of our thoughts. MITRE management support, particularly from Edward Lafferty and Judy Clapp, was also crucial.

A number of individuals were involved in the series of KNOBS implementations. The originator and leader of the KNOBS project was Carl Engelman. Since the inception of the project, a number of people have made substantial contributions to the system, and in some cases the main credit for existing facilities can be assigned. C. Engelman was responsible for the frame/rule architecture, the initial inference system, and the idea of constraint priorities and trapping. C. H. Berg put together the data base. E. A. Scarl designed the timeliness architecture and the first English parser. W. Stanton put in the rule-to-English translation, the rule editing, and improved the back-chaining inference mechanism. M. J. Pazzani built the English parser and dialogue system, and the frame swapper. J. K. Millen added the rule explanation facility. P. Andreae installed filters, generators, and automatic planning. F. Jernigan revamped the handling of time, and he and T. Fawcett provided the configuration management system. T. Fawcett contributed the display interface, truth maintenance and forward chaining. Others who contributed to various parts of the system are M. Bischoff, L. Ericson, W. A. Frawley, S. Hsu, I. C. Price, and S. Sundaram.

Special mention must be made of M. J. Pazzani, who is primarily responsible for the natural language interface in KNOBS. Much of the material in this document relating to the natural language understanding subsystem is an adaptation of his original writings. The criticisms of the *conceptual dependency* approach to natural language parsing and generation are due to the other authors.

Finally, this document has three authors. Each of the listed authors contributed a significant portion of this document, although no section is free of the editorial efforts of at least one other author. As a general observation, there is probably no topic on which all four authors completely agree. Mathematically speaking, the opinions herein are a subset of the union rather than the intersection of the individual opinions of the authors.

This document does not necessarily reflect opinions of The MITRE Corporation, nor does it reflect the opinions of Rome Air Development Center, the organization within the United States Air Force which sponsored this work.



Accession For	
NTIS GRA&I	<input checked="checked" type="checkbox"/>
DTIC TAB	<input type="checkbox"/>
Unannounced	<input type="checkbox"/>
Justification	
By	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
A-1	

KNOBS

The Final Report (1982)

WHENCE COMETH KNOBS?	1
KNOBS WAS AN EXPERIMENTAL TESTBED	1
THE APPLICATION	3
Stereotypical Planning	3
A Surprising Result	3
CRITICAL DECISIONS	3
Critical Decisions in the Planner System	4
Critical Decisions in the Natural Language System	5
Choosing a Universal Knowledge Representation Scheme ...	6
THE INITIAL RULE SYSTEM	7
MICROKNOBS	7
Retrospective: New ideas of MICROKNOBS	8
The Decision to Use Interlisp	8
Rules as the Only Representation of Knowledge	10
Backwards Chaining	10
Explanations	11
Changes in the Knowledge Base	12
THE INFERENCE MECHANISM	12
Consideration of the MICROKNOBS Rule Format	13
The INFER Function	14

Ordering Rules	14
Audit Trails	14
The Need for Indexing	15
New KNOBS Clause Format	15
FRAMES FOR DATA BASE ORGANIZATION	16
Frames Support Inheritance	16
Backward-chained Inheritance Through Multiple Paths	18
AKO and Other Relationships	19
THE KNOBS DATA BASE	20
FRL	20
FRL Functions	24
Inheritance	24
Procedural Attachment	24
THE FRL DECISION	25
THE EFFECT OF FRAMES ON INFERENCE	27
The Format of the Rule Evolved	27
Multiple Values	28
Inheritance	29
MISSION PLANNING -- THE PLANNING TASK	30
Stereotypical Planning	30
Multiple-frame Planning	31
Resource Conflicts are Not Stereotypical	31
"Set Valued Slots" and Stereotypical Planning	32
Mixed Initiative	32
From MICROKNOBS to KNOBS	33

KNOBS USES CONSTRAINTS	34
Simple and Complex Constraints	34
The Generic Mission Frame and Template	35
CONSTRAINTS	35
Non-Rule Constraints	37
Rule-Based Constraints	37
Daemons	38
UnDaemons	38
Retrospective Review of Constraints	38
CONSTRAINT TIMELINESS AND PRIORITY	39
Constraint Priorities	40
Timeliness and Trapping	40
Aircraft Availability -- The Effects of Timeliness	41
CONSTRAINT TIMELINESS AND EVALUATION	42
TIMELINESS, TRAPPING, and PRIORITIES	42
Re-evaluating Constraints	44
CONSTRAINT ARGUMENTS	46
The "Timeliness" Specification Language	46
Discussion of the "Timeliness" Specifications	47
COMPLEX CONSTRAINTS	48
THE NATURAL LANGUAGE SUBSYSTEM	50
WHY NATURAL LANGUAGE?	50
"Menus" Were Not Available	51
The Hypothetical ENUMERATE Command	51
The Need for Question Patterns	52

The Need for Ellipsis and Anaphora	52
The Need for a "Natural Language" Interface Has Nothing to do With "Natural Language"	53
THE MICROKNOBS PATTERN MATCHER	53
Discussion of the Pattern Matcher	54
THE ATN PARSER	55
Discussion of ATN-based Natural Language	56
THE CONCEPTUAL DEPENDENCY PARSER	57
Concepts	57
The Parsing Procedure	58
Dictionary Definitions	60
Anaphoric References	60
Arbitrary decisions in Conceptual Dependency	60
THE RESPONDER	61
Question-Answering	62
Inferences	62
Applications of CD-based Natural Language	64
Rule Editing	64
Truth Maintenance	65
Dialogue-Based Planning	65
TRANSLATION OF RULES TO ENGLISH	66
Translating LISP Expressions and Clauses	66
Condensation	67
EXPLANATIONS	68
The User's View	68

CHOICE GENERATION AND AUTOMATIC PLANNING	70
From Constraint Checking to Automatic Planning	70
KNOBS's Support Facilities	71
Does the Air Force want AUTOPLAN?	71
ENUMERATION	72
Implementation	73
Discussion of Enumeration	73
ORDERING	74
Discussion of SUGGEST-... ..	75
Implementation	76
Explanation	76
AUTOPLAN	76
Backtracking	77
Example	77
AUTOMATIC REPLANNING	79
The Forward-Chaining Rule Interpreter	79
Limitations of Replanning	81
CONCLUSIONS	82
Features in KNOBS	82
Critical Decisions and Insights	83
Weaknesses and Regrets	84
APPENDIX	86
A SAMPLE PLANNING SESSION	86
BIBLIOGRAPHY	92

Figures and Illustrations

1 Knobs pulls it all together	2
2 Development Environments	9
3 Inheritance	17
4 FRL Hierarchy	18
5 AIO and AKO Intertwine	20
6 Portion of the Data Base	21
7 Portion of the Target Data Base	22
8 Frames as Plans	34
9 Mission Frame Organization	36
10 The Initial Snapshot, DT Still Unbound	43
11 DT Given a Value, TIMEORDER Fails	44
12 If SERVICE Succeeds	45
13 Possible History for Figure 12	45
14 Relations Between a Mission, a Refueling, an Orbit, and a Service ..	49
15 From Surface Structure to Understanding	56
16 Part of the Hierarchy of Primitive Concepts	59
17 KNOBS Top-level Menu Display	86
18 Mission-level Menu Display	87

WHENCE COMETH KNOBS?

In mid-FY78 the Air Force Office of Scientific Research, Directorate of Mathematical and Information Sciences, commenced sponsorship at MITRE of a program to investigate and demonstrate the applicability of promising Artificial Intelligence technology to the automation of Air Force command and control mission planning and mission monitoring systems. The project came to be known informally as "KNOBS", an acronym for KNOwledge-Based System. The FY79 AFOSR funds were augmented by some from the Information Sciences Directorate of the Rome Air Development Center. Subsequently RADC took over support of the project through FY82, with supplementary funding from MITRE. This report is a final report on the four years FY79-FY82.

The intention of this document is to review as a case study the critical decisions that were made during the period 1978 to 1982 in the development of the KNOBS system from the perspective of continuing work in 1985. Some of the decisions led to fundamental new insights; others were good ideas at the time that just didn't work out as expected. This report is an attempt at a frank evaluation of both. The authors feel that the artificial intelligence literature does not contain enough forthright and honest evaluation of completed work, and hope that our modest contribution will encourage others to provide similar insight.

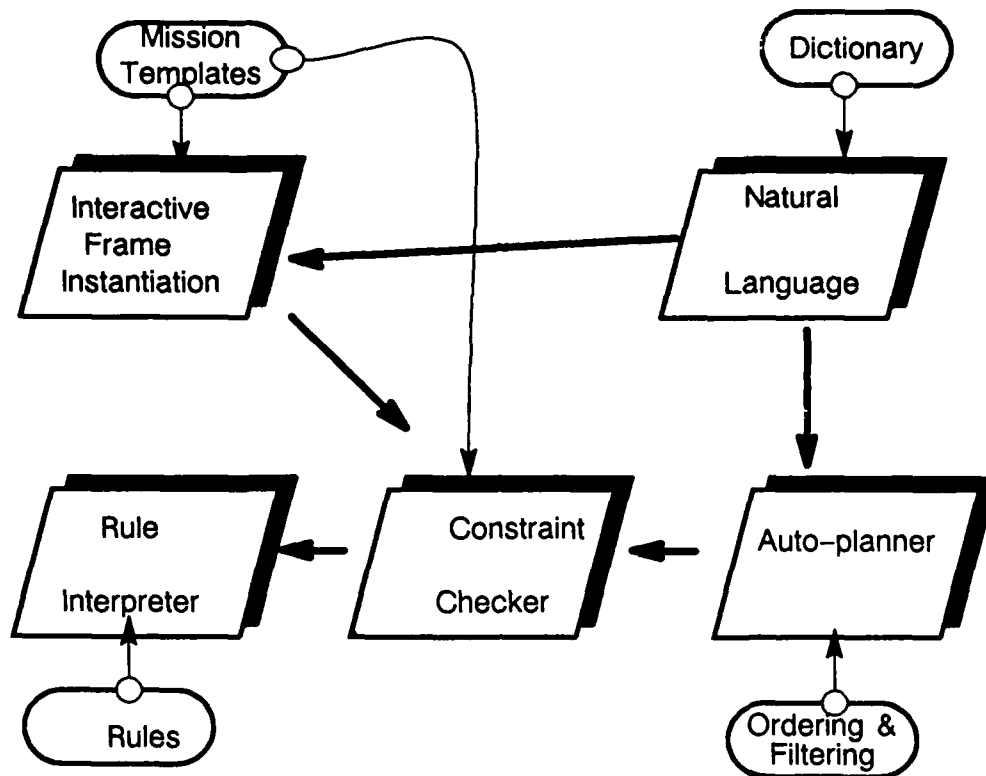
Among expert systems, KNOBS stands out because it unites several capabilities: frame representations for data, rule-based inference, English understanding using a conceptual-dependency parser for question-answering and control, and a constraint-based approach to planning. This complex architecture evolved from a simple rule-based system as a result of the functional and performance demands placed on the system.

Our review will be broken into two parts: the review of the planning system, and the review of the natural language system. For the period FY82 to FY85, the natural language system was partially developed under a separate RADC project with slightly different ends. Therefore, we will review only those portions of the natural language system that directly pertain to KNOBS.

KNOBS WAS AN EXPERIMENTAL TESTBED

Throughout this report, the reader should remember that KNOBS was never intended to be a software product, or even a prototype mission planning system. Rather, it was a testbed for ideas about how knowledge-based system technology and natural language processing technology could be combined to support the planning task.

KNOBS was a demonstration of capabilities meant to suggest what kinds of things *could* be accomplished. Most of the capabilities in KNOBS were incompletely implemented; the capabilities could be shown in one or two examples, but KNOBS seldom handled the general case. This partial implementation was appropriate for a demonstration of capability, but would



*Arrows indicate typical control flow
ovals are knowledge sources*

Figure 1: KNOBS Pulls It All Together

not be acceptable in a prototype (and certainly not in a production quality system).

As a testbed, KNOBS made only weak commitments to organizational and architectural conventions. In a certain sense, there is no KNOBS architecture. When a new type of mission is added to the planning repertoire, the inference mechanisms must usually be explicitly activated by attaching code to appropriate places in the data structure. When adding a new word to the vocabulary, code must be written to attach to various hooks in the dictionary. In order to add a new constraint, a new LISP function must be written that can, at its whim, invoke part or all of the rule-based inference mechanism.

This lack of commitment is acceptable (indeed, it is necessary) in an experimental testbed. How else can one experiment with, for example, how constraint checking should invoke the rule system? One would hope that at the conclusion of the experiments, we would be able to say what worked and what didn't, and then build a strong architecture in which the various inference mechanisms were automatically coordinated and invoked in a uniform manner. This report attempts to draw such conclusions.

THE APPLICATION

Stereotypical Planning

KNOBS was intended to be a framework in which knowledge-based planning aids could be constructed in a variety of domains, provided the application had a certain form: the planning had to be "stereotypical." Planning is "stereotypical" if it can be viewed as filling out a form with a number of blanks, where each blank is filled with the name of something -- either the name of a facility, a type of resource, a time, a quantity, a location, or a specific object in the data base.

Each form corresponds to a "plan element." If plan elements are used to fill blanks in other plan elements, then the planning problem becomes less stereotypical. If sets of things fill blanks, then the planning problem becomes less stereotypical.

Most of the inference mechanisms in KNOBS are built around the notion of a frame with slots. Frames can therefore represent plan elements, with "blanks" being represented by "slots."

A Surprising Result

Stereotypical planning is a far simpler type of planning than usually discussed in the artificial intelligence literature. Sequences of actions, and coordinated actions occurring in parallel are both outside the purview of stereotypical planning. Furthermore, the exclusion of planning sets of things is a great simplification.

It is surprising, therefore, that any real planning problem can be recast into this very narrow stereotypical planning format. KNOBS pushed stereotypical planning about as far as it can go. A follow-on program called KRS -- standing for "KNOBS Replanning System" (from FY83 to FY85) -- attempted to break the KNOBS approach out of the stereotypical planning regime to handle multiple coordinated activities; it met with mixed success and showed a negative result: the techniques that work for stereotypical planning do not generalize. This report will occasionally suggest where these techniques begin to break down.

CRITICAL DECISIONS

One would like to believe that critical decisions are made only after very careful consideration of all alternatives, and that once made the criteria be carefully documented. Real exploratory investigation does not work that way.

The KNOBS experience suggests that only in retrospect can a decision be recognized as having been crucial. Conversely and perversely, those decisions thought to be critical *at the time they were being made* frequently turn out not to matter at all. As a case in point, KRS (the follow-on to KNOBS) spent almost a year debating the "right" way to organize various plan elements to represent a refueling mission's interaction with an offensive counter-air mission

(the type of mission KNOBS plans). In retrospect, the particular organization of frames didn't really matter.

This report points out, from a distant perspective, what critical decisions were made and what their consequences were. The reader is cautioned not to interpret the comments in this report as reflecting the considerations of the KNOBS researchers and implementors at the time (FY79-FY81). Rather, the authors arrived at their various opinions based on work done on KNOBS and other systems developed at MITRE and elsewhere, including systems to be built in FY86 and beyond.

Most of the discussion will be postponed until the relevant portion of the report. In the immediately following paragraphs, we will point out the kinds of issues that are critical, and alert the reader to some of the more important themes of the KNOBS research.

CRITICAL DECISIONS IN THE PLANNER SYSTEM

KNOBS uses rules. Initially, in the MICROKNOBS incarnation of FY78, it only used rules, and only used them in a backward-chaining manner. By the end of the period (in late 1982), KNOBS used several different forms of inference, only one of which was rule-based. The others -- notably frame-based inheritance and constraint-checking -- represent alternative methods of capturing knowledge in a computer program. The emphasis on constraints as a *separate entity* is one of the great insights implemented in the KNOBS system. Most of the critical decisions we will examine concern these changes.

Having decided to use FRL-style frames to represent the things involved in planning (e.g., the types of resources, the facilities, and the plan elements), and having decided on some format for expressing some knowledge as rules, attaching other knowledge to the FRL frame hierarchy, and representing still other knowledge as constraints, another set of critical decisions must be made: when a constraint succeeds or fails, what information should be returned? In a slightly different guise, when attempting to infer something using the rule system, what information should be returned on a successful or unsuccessful attempt?

One of the critical decisions in the planning system was that KNOBS should both support a human planner and be able to automatically plan on its own. That is, KNOBS could complete a partial plan from any stage, provided that a consistent continuation exists. From this decision a number of secondary decisions had to be made; a small sample follows:

How should KNOBS distinguish its decisions from the users?

How should candidates for filling a slot be enumerated? (It turns out that the "right" answer for interactive purposes may be exactly the "wrong" answer for autoplanning.)

Should any information about why a constraint failed be fed back into the enumeration? If so, what information, and by what route?

Should all constraints be checked, or only a few that can fail for "different" reasons?

Even though the KNOBS project was to explore knowledge-based techniques, the developers did not choose to enforce a rule-oriented representation of knowledge. Indeed, the second-worst critical decision in the entire planning system concerned how rules should be invoked; it was a decision made by default. Nobody decided that all activations of the rule system should be through application-dependent functions; rather, the implementors simply did it that way. This is why, for example, that constraint checking invokes the rule-based inference mechanisms only if the constraint function (which is always handcrafted) happens to invoke it. Generally speaking, the implementation of KNOBS decides on which inference mechanisms should be invoked when in an *ad hoc* and often arbitrary manner.

CRITICAL DECISIONS IN THE NATURAL LANGUAGE SYSTEM

It is in the natural language system that we find the single worst critical decision of the entire KNOBS project: that the natural language system should be intimately intertwined with the planning system. As an example, in checking the constraints applicable to some value for a slot, KNOBS used a scheme (called "trapping") to only try a particular subset of the applicable constraints. This subset was the one felt "relevant" to the user, *not* the subset that would provide the best corrective information to the slot-filling enumerators. Thus, the needs of the interface conflict with the needs of the inference mechanisms. Rather than resolving this conflict by separating the interface from the inference mechanisms (and supporting both), KNOBS only supported the interface.

We'll see that the selection of which constraint failures to tell the user about is very easy to make after all have run (provided they return the proper information when they fail). Deciding which constraint to run ahead of time, on the other hand, is a very complex decision. Because the user interaction considerations were so intermingled with the planning considerations (the constraints generate failure explanations while they are running), the KNOBS implementation was encouraged to go down the hard and hairy path: about two man-years went into building and modifying the constraint trapping mechanisms.

If constraints were to return some useful information when they fail (for example, when a constraint concerning a time failed, it could return the time interval it would find acceptable), then we would clearly want to run all the constraints so that we would have the most insight into how planning should proceed. This style of reasoning could not be investigated because the natural language system has preempted the planning mechanisms for its own purposes, as explained above and in more detail later in this report.

Choosing a Universal Knowledge Representation Scheme

Since the goal of the natural language system is to accept arbitrary sentences from the user, it is important that the language system's internal representation of the meaning of a user's input in fact be able to represent any message the user might wish to convey. Naturally, any internal representation of meaning is going to be only an approximation of what the user meant. But there should be some principles guiding how that internal representation encodes meaning.

The knowledge representation scheme eventually chosen for the natural language system was that of "Conceptual Dependency." While all knowledge representation schemes are subject to many arbitrary and *ad hoc* decisions, conceptual dependency "theory" is particularly bad in this respect, see discussion in this report's section on "arbitrary decisions in conceptual dependency."

THE INITIAL RULE SYSTEM

KNOBS is a knowledge-based system, meaning that it possesses factual knowledge in some specialized domain, and that it uses that knowledge in a problem-solving capacity to assist a human specialist. The term "knowledge-based system" is used to describe both so-called "expert" systems and more general types of systems that use a variety of inference mechanisms. Chess-playing programs are not "knowledge-based" programs, because they have almost no knowledge that one would call "factual", aside from some standard openings: their heuristics and position evaluation *knowledge* is encoded in computer code. Information retrieval systems have factual knowledge but no problem-solving ability. One of the best known expert systems is MYCIN [SHOR76], whose purpose is to aid physicians in the diagnosis and treatment of bacterial infections. One of the first was DENDRAL, applied to aid chemists in the analysis of the molecular structure of organic samples [FEIG71].

KNOBS is a knowledge-based system which captures the knowledge of human experts in a variety of information structures, including so-called rules. By using knowledge-based programming techniques, several advantages over more conventional software were anticipated:

- o New information could be incorporated rapidly - not just situation reports, but changes in the rules for acting on them.
- o The system could take an active role in planning, identifying problems and suggesting alternatives. It could explain its reasoning.
- o English could be used to access, modify, and control the system - not exclusively, but whenever an abbreviated command interface is not expressive enough.

KNOBS did not spring into being over night. One of the early decisions of the project was that a functional capability needed to be demonstrated very early. The early drive towards a demonstration system resulted in MICROKNOBS.

MICROKNOBS

At the conclusion of FY79 an initial demonstration system, called "MICROKNOBS", had been constructed, though at a level of complexity and realism considerably below what would eventually be attained. It was implemented in Interlisp, a version of LISP initially devised at BBN and transported to a number of different computers, including Digital Equipment Corporation PDP-10 and DECSystem-20 computers. For a first application in an Air Force context, MICROKNOBS was given some rudimentary information about aircraft types, target types, weather, munitions, and electronic countermeasures; this information was used to generate recommendations for munitions selection.

MICROKNOBS embodied some of the principles that guided later development in KNOBS. The general characteristics and organization of MICROKNOBS are recognizable in KNOBS, although the code has been substantially revised: the inference system, the natural language interface (albeit MICROKNOBS used what was essentially a keyword-driven pattern matcher), the data base, explanations, and rule editing. The principal differences between KNOBS and MICROKNOBS are in KNOBS's greatly expanded and reorganized data base (represented in *FRL frames* -- see discussion below), in the change of emphasis from *rules* to *constraints*, the new facilities for planning complex missions, and the advanced natural language subsystem. A display-oriented user interface made a difference as well, both externally and in the top-level structure of KNOBS.

Retrospective: New Ideas of MICROKNOBS

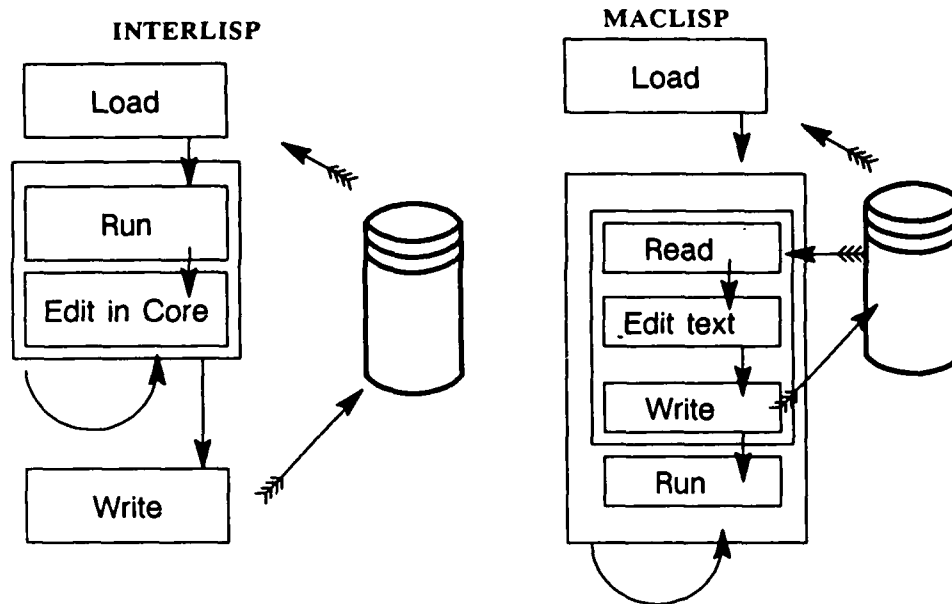
It is somewhat difficult to appreciate the new insights of MICROKNOBS today because several of its new ideas are well-known today. For example, when MICROKNOBS was being developed, having the inference engine automatically maintain justifications was a somewhat new idea; today it is standard. As another example, using declarative rules as the means for capturing knowledge about planning was a goal seldom achieved in an implementation; today it is common practice (at least in the reports written about implementations).

One of the outstanding ideas in MICROKNOBS that is fresh even today is that the rule base should be easily altered by the user while the system is being used. Even more significantly, we'll see that KNOBS's English component supports some rule changes using natural language as the medium by which the new rule is added.

The Decision to Use Interlisp

The decision to use Interlisp (used primarily on the west coast and at BBN in Cambridge, MA) rather than MacLisp (used at MIT, CMU, and other east coast centers) turns out to have been a mistake, due both to the later availability of specialized hardware for fast LISP processing based on MacLisp, and the 1985 emergence of *Common Lisp* -- a derivative of MacLisp -- as a proposed LISP standard. We can only speculate as to why this choice was made: MacLisp had support only under MIT's ITS operating system and under the MULTICS system, while Interlisp ran under DEC's standard operating system. MULTICS time was expensive. Therefore, unless one could dedicate a PDP-10 to AI work, MacLisp was not an option. Furthermore, computer time for running Interlisp could be purchased both at BBN and at USC on the west coast. The latter was particularly important since demonstrations could be given on that machine in the early morning (say 9:00 EST) before the west coast users were awake (6:00 PST).

It turned out that this very early decision made essentially for short-term economics had an unexpected and high cost. In the Interlisp programming environment, one edits the internal representation of code. The basic cycle is one of



Interlisp vs. MacLisp Program Development

Figure 2: Development Environments

"load-run-edit-run ... edit-run-write"

MacLisp (and subsequent dialects, including ZetaLisp which runs on LISP Machines and Common LISP which will be the standard definition of the core of LISP) follows a more conventional cycle of

"load-run-read-edit-write-run- ... read-edit-write-run"

The difference is that with MacLISP one edits text (albeit with a text editor that "knows, understands, and loves" LISP), while in Interlisp, one edits live structure in memory. The effect of this is that with MacLISP one doesn't pay any run-time space cost for comments, while one does with Interlisp (even though Interlisp comments take no space in compiled code, they do in interpreted code). Furthermore, comments in MacLISP could be put anywhere (since they were ignored by the LISP reader), while in Interlisp care had to be taken. The net result of this early decision is that KNOBS code is very poorly commented while other code written by some of the same individuals in a MacLISP environment is well commented (at least by AI research system standards).

Let me emphasize: the KNOBS experience leads to the conclusion that a "live structure" editor -- as found in InterLisp -- discourages adequate commenting of code. A consequence of this observation is that any knowledge-based system that allows rules to be changed while the system is running must take care to facilitate documentation of those changes.

Rules as the Only Representation of Knowledge

MICROKNOBS was strictly a rule-based system. Its knowledge was in the form of simple implications of the general form "if the hypotheses H1, H2, ..., Hn are satisfied, then conclude C1, C2, ..., Cn." Information needed to check the hypotheses of a rule could come from the data base, from the conclusion of another rule, or from some arbitrary LISP code.

A typical rule for the selection of munitions, as printed by MICROKNOBS, is:

IF:

1: THE target IS HARDER THAN HARD

AND 2: THE CLOUD COVER OF THE target IS LESS THAN 1

AND 3: THE target IS LESS DEFENDED THAN LIGHTLY DEFENDED

THEN:

1: THE BEST MUNITIONS FOR THE target IS LASER GUIDED

Lower case indicates variables. The rule is actually stored internally in a list-structured form suitable for Interlisp processing, but let us focus for the moment on the logical operation of the system.

A rule like the one above would be invoked by an English query of the form, "CHOOSE MUNITIONS FOR TARGET 3". For natural language interaction with users, MICROKNOBS had a surprisingly powerful syntactic pattern matcher with synonym and Interlisp spelling correction capabilities. It would accept queries about anything in the data base, such as: "WHAT IS THE HARDNESS OF TARGET 3", as well as requests for munitions selection. It could resolve certain pronominal references, such as "CHOOSE MUNITIONS FOR IT", if, as in this case, a previous query had mentioned a suitable referent like "TARGET 3".

The cloud cover, hardness, and other facts about targets were stored in a data base. A data base entry is in the form of the conclusion of a rule, like "THE CLOUD COVER OF TARGET 1 IS 0". In the example rule above, there is a hypothesis "THE CLOUD COVER OF THE target IS LESS THAN 1"; checking it requires both a data base lookup and an arithmetic comparison, which the inference mechanism takes in stride.

Backward Chaining

The hardness of the target is an example of a fact that must be deduced from another rule. MICROKNOBS has to look for a rule with a conclusion about hardness, such as:

IF:

1: THE target IS A BRIDGE

THEN:

1: THE target IS COLD

AND 2: THE target IS VERY HARD

AND 3: THE target IS STATIONARY

Then, when checking the hypotheses of this rule, it will look for an entry in the data base to find out whether or not the current target is a bridge.

The pattern of reasoning illustrated above proceeds from "conclusion" desired to hypotheses which, if true, would imply the conclusion. When the conclusion of a rule is needed, the system sets up the hypotheses of that rule as new goals for deduction. This process is characteristic of a backward-chaining inference mechanism. An alternative to backward-chaining inference is so-called "forward-chaining" which proceeds from hypothesis to conclusion.

Roughly speaking, *backward-chaining* is goal-driven reasoning, rather than data-driven reasoning. In a backward-chaining system, the rules are invoked to answer specific questions rather than to warn or alert the system. The distinction between *backward chaining* and the alternative of *forward-chaining* can become blurred in sophisticated rule-based systems.

The decision to use backward-chaining exclusively appears to have been basically correct. There are two exceptions: first, when gathering recommendations the rules that map features to recommendations should be forward-chained. Secondly, when told that, for example, "Hahn is an airbase," inferences specializing general knowledge about airbases to particular knowledge about Hahn should be forward-chained. We will return to these exceptions for a detailed examination later in this report (see section on "The FRL decision").

Explanations

A rule-based system satisfied an important goal for an expert system: that the functioning of the program could be controlled by strategies easily understood by users. Not only can the system print out any rules in English, but it can also indicate which rules played a part in any particular deduction, and how they were used.

Explanations in MICROKNOBS were prompted by the query, "WHY?" after the system stated a conclusion. When laser guided munitions are chosen for target 1, for example, the explanation is:

BY MR1: THE BEST MUNITIONS FOR TARGET 1 IS LASER GUIDED
SINCE:

1: BY TR3: TARGET 1 IS VERY HARD

SINCE:

1: DATA: TARGET 1 IS A BRIDGE

2: DATA: THE CLOUD COVER OF TARGET 1 IS 0

3: DATA: TARGET 1 IS LIGHTLY DEFENDED

The explanation names the rule that delivered the conclusion and indicates how its hypotheses were satisfied. If back-chaining was needed to establish a hypothesis, the rule needed is named and given the same treatment, recursively.

Changes in the Knowledge Base

MICROKNOBS could add or delete rules and facts, and edit rules. When adding or editing a rule, the system would take the user through the hypotheses of the rule one at a time; new hypotheses or conclusions were entered in near-English. The user was, of course, limited to a highly stylized syntax.

THE INFERENCE MECHANISM

The internal form of rules, and the operation of the back-chaining inference in KNOBS has its genesis in MICROKNOBS. The aspects of the implementation that changed, and the reasons for the changes, will be covered in later sections.

A MICROKNOBS rule had the form:

(rulename (hypothesis*) (clause+))

where "*" indicates zero or more occurrences of the term it follows, and "+" indicates one or more occurrences. The second element of the rule is a list of hypotheses, and the third is a non-empty list of conclusions. A hypothesis is a list that begins with a clause and may be followed by some side conditions:

(clause side-condition*)

Clauses used in hypotheses or conclusions were sentences that looked almost like the way they were printed out. Some examples are:

(?TARGET IS-A AIRBASE)

(THE DEFENSE OF ?TARGET IS HIGHLY-DEFENDED)

An identifier beginning with a question mark "?" is a variable. When printing rules out, the system replaced the variable "?TARGET" by "THE target", eliminated hyphens in "IS-A" and other keywords, and performed some other transformations as well.

Side conditions were essentially LISP expressions representing function calls. For example, the side condition

(HARDER ?X HARD)

would, after a constant like SOFT had replaced the variable ?X, be evaluated in the form

(HARDER 'SOFT 'HARD)

when the hypothesis was checked. An interesting feature of the transformation of rules to their printed form was the way a hypothesis with both a clause and side condition, like

((THE DEFENSE OF ?TARGET IS ?Z)
(LESS-DEFENDED ?Z LIGHTLY-DEFENDED))

was condensed into a single statement,

THE DEFENSE OF THE TARGET IS LESS
DEFENDED THAN LIGHTLY-DEFENDED.

The mechanism for doing this, which was generalized to handle much less straightforward cases, and produce better English, will be described in Section 5.

Consideration of the MICROKNOBS Rule Format

The format of rules just described has two related features which should be examined in more detail. The first clause in the hypothesis list received special attention in that it contained the first reference of every variable that needed to be bound. The pattern matcher needed to bind all these variables while looking for a single clause in the list of currently believed facts (we'll sometimes use the term "fact base" to refer to this list). Only then could the side conditions be evaluated as predicates. The predicate's LISP code could then invoke the inference mechanisms, either to confirm the side-condition or to refute it.

Consider the problem of finding a country's airbase X that has aircraft Y when the kinds of facts in the fact base are:

(HAS-UNIT AIRBASE-HAHN TFW-109)
(HAS-AIRCRAFT TFW-109 F-4C).

We could not have a rule

(RULE123
((IN ?X ?COUNTRY) (HAS-UNIT ?X ?Y) (HAS-AIRCRAFT ?Y ?Z))
((HAS-AIRCRAFT ?COUNTRY ?Z)))

because the two side conditions "HAS-UNIT" and "HAS-AIRCRAFT" refer to a variable "?Y" that has not been bound by the conclusion or the first clause in the hypothesis. Putting the "HAS-AIRCRAFT" clause first leaves "?X" unbound. So the only choice is to put the "HAS-UNIT" clause first, even though that is computationally expensive due to the number of airbase/unit pairs to be enumerated. Note that the problem just mentioned can be solved by allowing multiple clauses as well as having side-conditions.

Furthermore, the fact that the side-conditions were interpreted as functions set a dangerous precedent of capturing some knowledge in procedures. The knowledge engineer always had a choice: either write some rules to infer some condition, or write some code to compute whether some condition was true or not.

The INFER function

INFER is the back-chaining inference function. Its call is in the form:

(INFER clause)

with some additional optional arguments to control the search. The "clause" argument is the conclusion of some rule. In the example above, looking for the best munitions for target 1, the call was:

(INFER '(THE BEST-MUNITIONS FOR TARGET-1 IS ?X))

INFER attempts to establish the requested conclusion by finding a fact or rule whose conclusion matches this one, substituting TARGET-1 if a variable ?TARGET appears in the corresponding spot. If it was the conclusion of a rule, INFER attempts to establish the hypotheses of the rule. Each hypothesis in the rule generates a recursive call on INFER.

If INFER succeeds, it will, in the process, find a value for each variable in the clause, in this case the munitions ?X. If it fails, another rule is sought. If no more applicable rules are found, INFER fails, returning NIL.

Ordering Rules

The order of rules is significant, because some rules are more specific than others. A rule is more specific than a second if the satisfaction of its hypotheses implies the satisfaction of the second's hypotheses. A more specific rule should appear earlier in the ordering, since any less specific rule preceding it would always preempt it.

Audit Trails

If INFER is successful, it returns not only a value for each variable in the conclusion, but also an "audit trail" that contains the information necessary to answer the question "WHY?".

The value returned by INFER is of the form:

(((variable . value)*) audit-trail)

where the audit trail is of the form

(DATA clause)

or

(rulename clause audit-trail*).

In both forms of the audit trail, the clause has constants filled in for all variables. In the first form, the keyword DATA indicates that the inference succeeded by finding a fact. In the second form, the clause is the conclusion of the named rule, and the nested audit trails are from the hypotheses of that rule.

Notice that MICROKNOBS simply returned NIL if INFER failed to show some conclusion. This is an error: inference mechanisms should distinguish between failing to prove or disprove a clause, and failing to prove because the clause is known to be false. Far better is to admit that any function that "proves" something needs to return one of three values: *true*, *false*, or *don't know*. The *don't know* response could result either from having tried to show both *true* and *false* and failing on both, or from simply giving up on trying to show either. MICROKNOBS is not the only system to make this mistake. Micro-PLANNER [SUSS 70] and PROLOG [CLME 81] both interpret failure as negation.

The Need for Indexing

Rules were stored as elements of lists, called rulesets; this meant that a linear search over a ruleset was necessary to find a rule matching a desired conclusion.

There were two rulesets: one for conclusions about best munitions, and the other for conclusions about targets. The fact base was also stored in a single list, which had to be searched linearly.

When the data base and number of rules were expanded, it was anticipated that searching for facts and rules would become excessively time-consuming unless appropriate forms of indexing were introduced. The frame system adopted for the data base provided its own means for indexing facts; this will be discussed in the next section. The need to index into the collection of rules led to a change in the rule format for KNOBS.

New KNOBS Clause Format

In MICROKNOBS the first clause also had a very English-like word order format: the clause

(THE DEFENSE OF ?TARGET IS ?Z)

does not have the typical LISP organization of

(DEFENDS ?Z ?TARGET).

When reimplementing the inference mechanism in the present KNOBS system, all clauses were standardized to look like

(relation-name argument1 argument2...)

where each argument is either a constant or a variable.

This construction uses the property list of the relation-name as a place to hold information needed by INFER, and also for information needed by the printing functions. If a clause with a given relation name can be established by the conclusion of some rule, then the relation name has a BCINDEX property, whose value is a list of all such rules, grouped by ruleset.

FRAMES FOR DATA BASE ORGANIZATION

The MICROKNOBS data base was simply a list of clauses, which had to be searched linearly for facts matching a desired conclusion. The need to search the data base more efficiently was one reason for introducing a frame system. The other reason was the naturalness with which frames could be used to support mission planning; that topic will be discussed in Section 3.

The idea of frames as a knowledge representation technique arose from an article by Minsky [MINS75]. Frames are a generalization of property lists. As with a simple property list, frames store attributes of a named object. If one wishes to find the hardness of a particular target, for example, it is natural to place that information under the HARDNESS property of the target name. Finding this property still requires a linear search of all the properties of the target, but the code can take advantage of the efficient built-in LISP mechanism for locating property lists of atoms. (The actual implementation of frames used in KNOBS includes the entire frame, with all attributes, under the FRAME property of the LISP atom naming the frame.)

Two central ideas separate "frames" from simple property lists: first, frames can "inherit" knowledge from a more general frame; second, there are various "daemons" associated with properties. The mechanism for attaching "daemons" to properties -- called a "facet" of a "slot" in a frame -- was also used to attach other kinds of auxiliary information about the value in a slot. These ideas will be presented in detail below.

Frames Support Inheritance

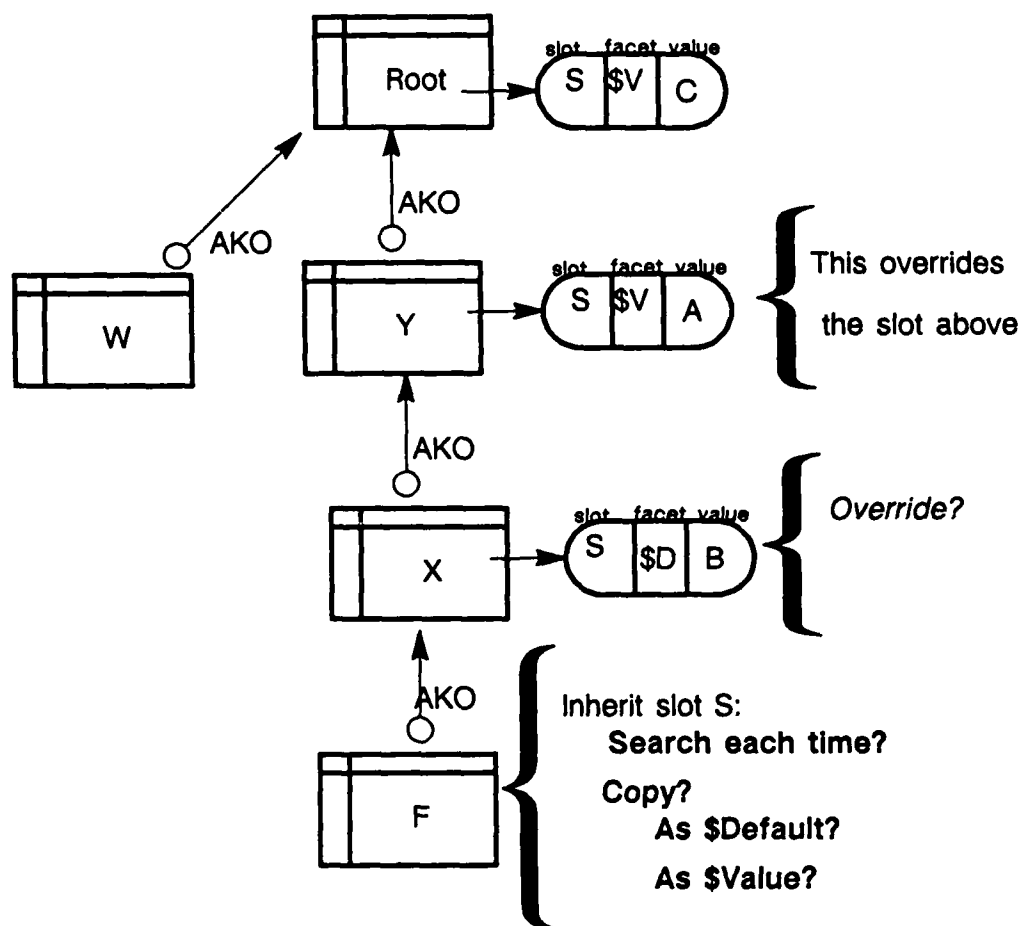
Frames can be related hierarchically for more economical storage of information. KNOBS has individual frames for each aircraft type, for example, but there is a default maximum speed for all aircraft. The default speed is placed in the SPEED attribute of a generic AIRCRAFT frame, where it can be inherited by individual aircraft types that do not have their own characteristic maximum speed.

Inheritance affects the value of a slot *S* in a frame *F* as described below. The chain of superiors of a frame *F* relative to a designated set of "inheritance" slots (let's assume there is only one inheritance slot named AKO) is the ordered set of frames enumerated by

AKO(*F*), AKO(AKO(*F*)), ... , frame-at-root.

A slot with both a "default" value and an "assigned" value is considered to have the value "assigned" rather than the "default." A slot without a value (either assigned or default) is considered to have an inherited value equal to the value of the slot *S* of the first frame (call it *I*) in *F*'s chain of superiors whose slot *S* has a value. The inherited value is copied into *F*'s slot *S*, either as the "assigned" or "default" value depending on the value's status in frame *I*.

Note that the scheme above gives rise to *two forms* of default behavior. Suppose *X* is AKO *Y*. *Y*'s *S* slot can have a value *A* (residing in the \$VALUE facet of the slot). In the absence of other information, then *X*'s *S* slot will then



Inheritance and Defaults — Unresolved Design Issues

Figure 3: Inheritance

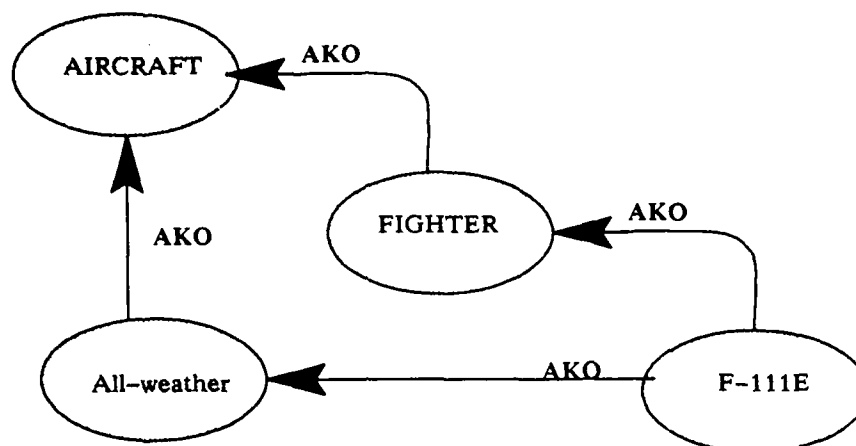
default to A. On the other hand, one could assign a value B to X's S slot in the \$DEFAULT facet (that is, a value with the status of being a default). KNOBS usually used the \$VALUE facet. The presence of these two almost identical notions of "default" is a weakness of the FRL implementation of the notion of frames.

A further design decision in the implementation of inheritance involves whether inherited values should in fact be copied into the inheriting frame, or should the inheritance be redone each time the value is needed. The efficient solution is, of course, to copy; KNOBS usually did NOT copy, the intention being to have the frames change with time. However, the KNOBS data base is actually quite static (e.g., which airbase has what aircraft tends not to change during a planning session).

The usual justification for using inheritance is that of saving space. While the space savings are significant, we recognize that the savings result from a time/space trade-off: the space is saved by doing a fairly expensive lookup each time some property is needed. Philosophically, there is a much more important

reason for using frames: inside a knowledge-based system, a single "fact" should only be recorded in a single place. That is, we know the fact that "all airbases have control towers." This fact should only be stated once for the general notion of "airbase," not once for each particular airbase.

The hierarchy of object types is itself information that can be important for certain purposes. The fact that an aircraft is a fighter rather than a reconnaissance plane influences its choice for different kinds of missions. Thus, the frame hierarchy includes the path:



Part of the FRL Frame Hierarchy

Figure 4: FRL Hierarchy

Of course, the "fighter" role of an F-111E could be stored as another of its attributes, instead of creating a FIGHTER frame, but the latter choice allows us to use the same inheritance mechanism in a consistent way to answer all questions of the form "Is X a kind of Y?" The FIGHTER frame is also a good place to put generic information about fighters.

Intermediate frames, like the FIGHTER frame above, create complications in handling inheritance. The default speed of an F-111E, for example, now has to be inherited through two generations. Furthermore, another category of aircraft is "all-weather-aircraft", and the same aircraft can be both a fighter and an all-weather-aircraft. This means that attributes can be inherited via multiple routes or from multiple ancestors. The frame system is not actually a hierarchy, but an acyclic graph. The diagram above shows such an acyclic graph.

Backward-chained Inheritance Through Multiple Paths

Just as with the MICROKNOBS rules, the mode of reasoning used in frame-style inheritance is backward-chaining. To find if F-111E's have wings, knowing that

((HAS-PROPERTY ?Z ?X) (IS-A ?Y ?Z)) -> (HAS-PROPERTY ?Y ?X)

backward-chaining will pose the question "Do fighters have wings?" The same basic inheritance rule could be used again to ask if airplanes have wings, etc. until the inference mechanism finds some class for which the fact base already contains an assertion that everything in the class has the property, or until the inference mechanism gets to the top of the class hierarchy.

If the question to be answered by inheritance is infrequently asked, then it makes sense to save space and use the time to do the inheritance when it is needed. However, KNOBS answers the question "Does a specific radar radiate?" by inheritance virtually every time it plans a mission that flies near a SAM site. For frequently asked questions, it seems to make more sense to "forward-chain" inheritance to record for each individual radar that it does indeed radiate. Unfortunately, the decision on whether to inherit *as needed* or *on initialization* is usually a design decision; it would have been possible to select certain slots to inherit by copying (as discussed earlier), but such was not done in KNOBS.

AKO and Other Relationships

The normal route for inheriting attributes is through the relationship "a kind of", customarily abbreviated "AKO". AKO is a relationship between generic objects, or between categories of objects. Note that in the standard theory of *frames*, AKO is itself just another attribute, which happens to be filled by the name of a frame rather than a number or some other value.

The relationship between individuals and their categories is somewhat different. For example, Alconbury is an individual airbase, not a kind of airbase. Its relationship to the airbase frame is AIO (An Individual Of). The concept of individuals has occurred in the literature; they were referred to as "instances" in [STEF79] and [SZOL77].

The reason for making the distinction between AIO and AKO is brought out in the following popular example: Suppose Fred is an African elephant, which are elephants. Elephant is a species, but we cannot conclude that Fred is a species.

The solution here is that Fred is an individual elephant, and elephant is an individual species.

On the other hand, an elephant is a kind of animal, and it is legitimate to conclude that Fred is an (individual) animal. Thus, it makes sense to inherit an AIO relationship over a path that begins with an AIO link and continues with AKO links. Other attributes of the class can be inherited over such a path by the individual.

Other transitive relationships can be useful paths for inheritance. KNOBS rules sometimes ask, for example, whether a target has a part which is a radio emitter. Thus, "a part of" (APO) relationships are also found in the KNOBS data base.

Finally, there are frame relationships which are useful in themselves as data but do not participate in inheritance paths. An example is the "a group of" (AGO)

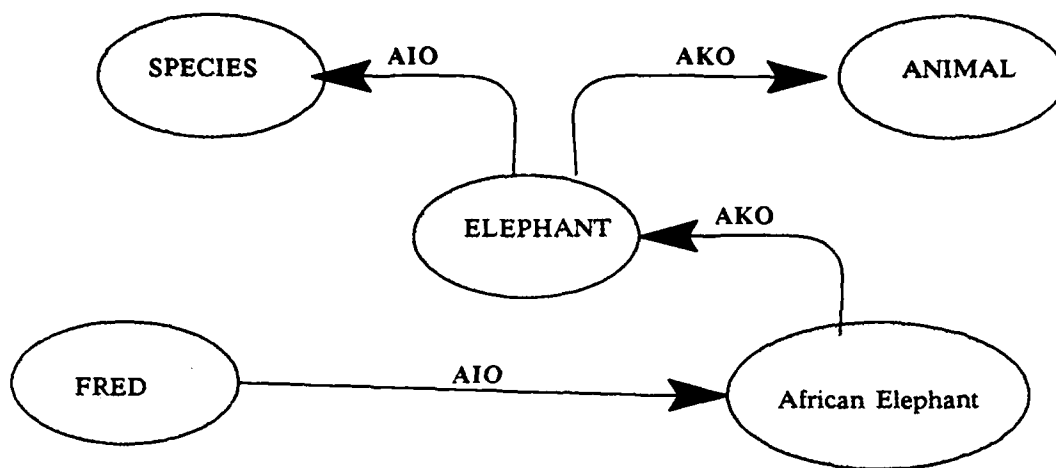


Figure 5: *AIO and AKO Intertwine*

relationship. This occurs in the KNOBS data base between the group of aircraft belonging to a particular combat unit and the generic aircraft frame.

These relationships all have inverses, so that they can be traced either upwards or downwards. Downward links are applied in KNOBS to generate choice sets. For example, a mission may require a fighter, so the inverse of the AKO link is followed to produce a list of fighter aircraft types.

THE KNOBS DATA BASE

The current KNOBS data base has information about 53 tactical units distributed among eight airbases in Germany and Great Britain, and about 700 targets. It has the speeds and ranges of eighteen types of aircraft, and the kinds of radar associated with several types of surface-to-air missiles (SAMs). The data in the experimental system is plausible, but neither classified nor accurate.

Some of the relationships in the KNOBS data base are shown in Figure 2.1. The figure hints at the size and depth of the data base, which contains about a thousand frames. The data base is so large, in fact, that much of it is kept in source files. Normally, a frame is kept under the FRAME property of the atom naming it. For most KNOBS frames, however, the FRAME property has, instead, the name of a file in which the full frame can be found. Functions that access frames were rewritten to recognize the file reference and load the frame when needed, using a facility called the "frame swapper." Only a few frames are locked into main memory.

FRL

It appeared that developing software to create and access frames, and handle inheritance properly, would be a significant task. Several frame representation languages or data access packages had been developed as LISP extensions;

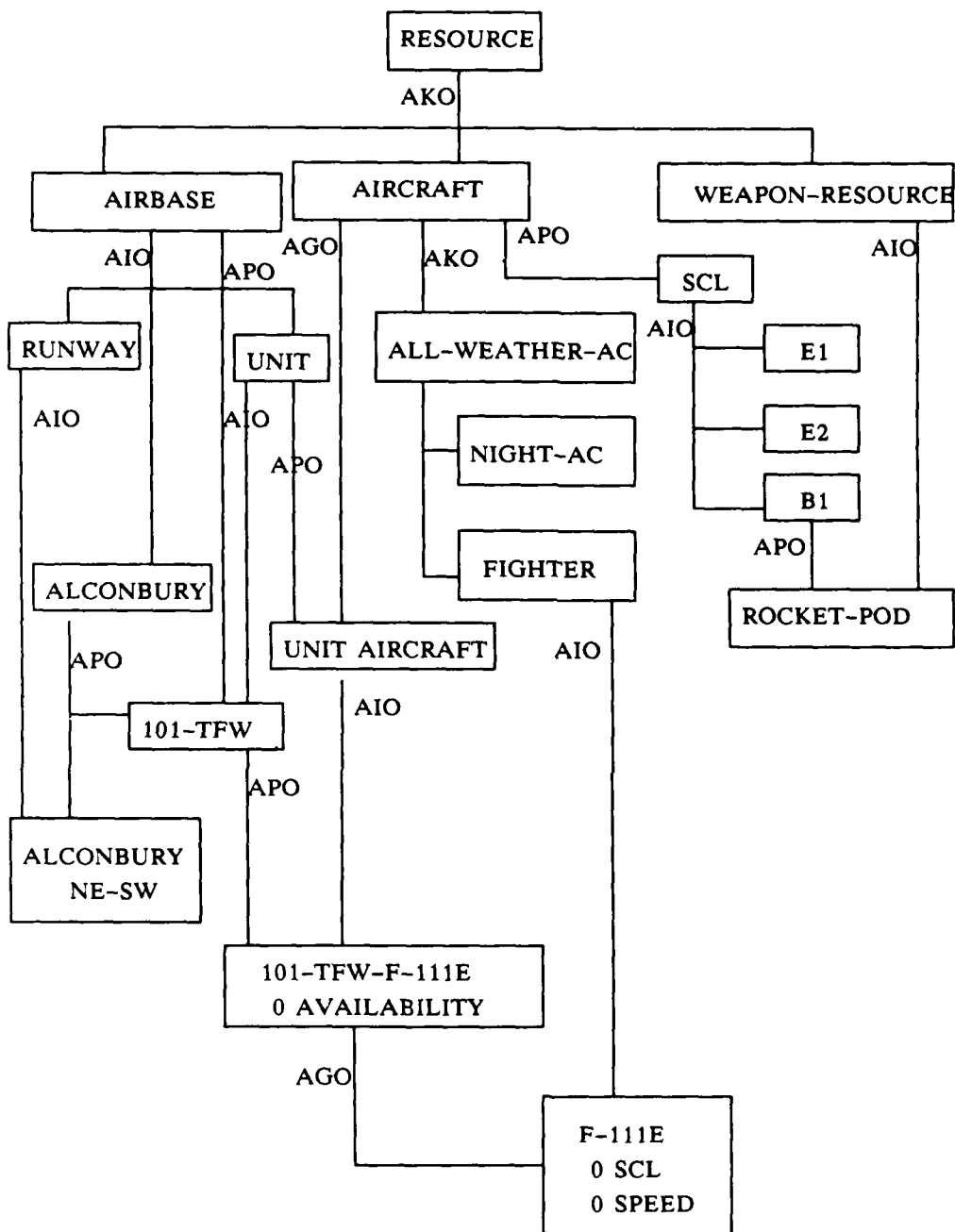


Figure 6:

A Sample of the Resource Portion of the Data Base. AIO (An Instance OF), APO (A Part Of), AKO (A Kind Of), and AGO (A Group Of) links are shown.

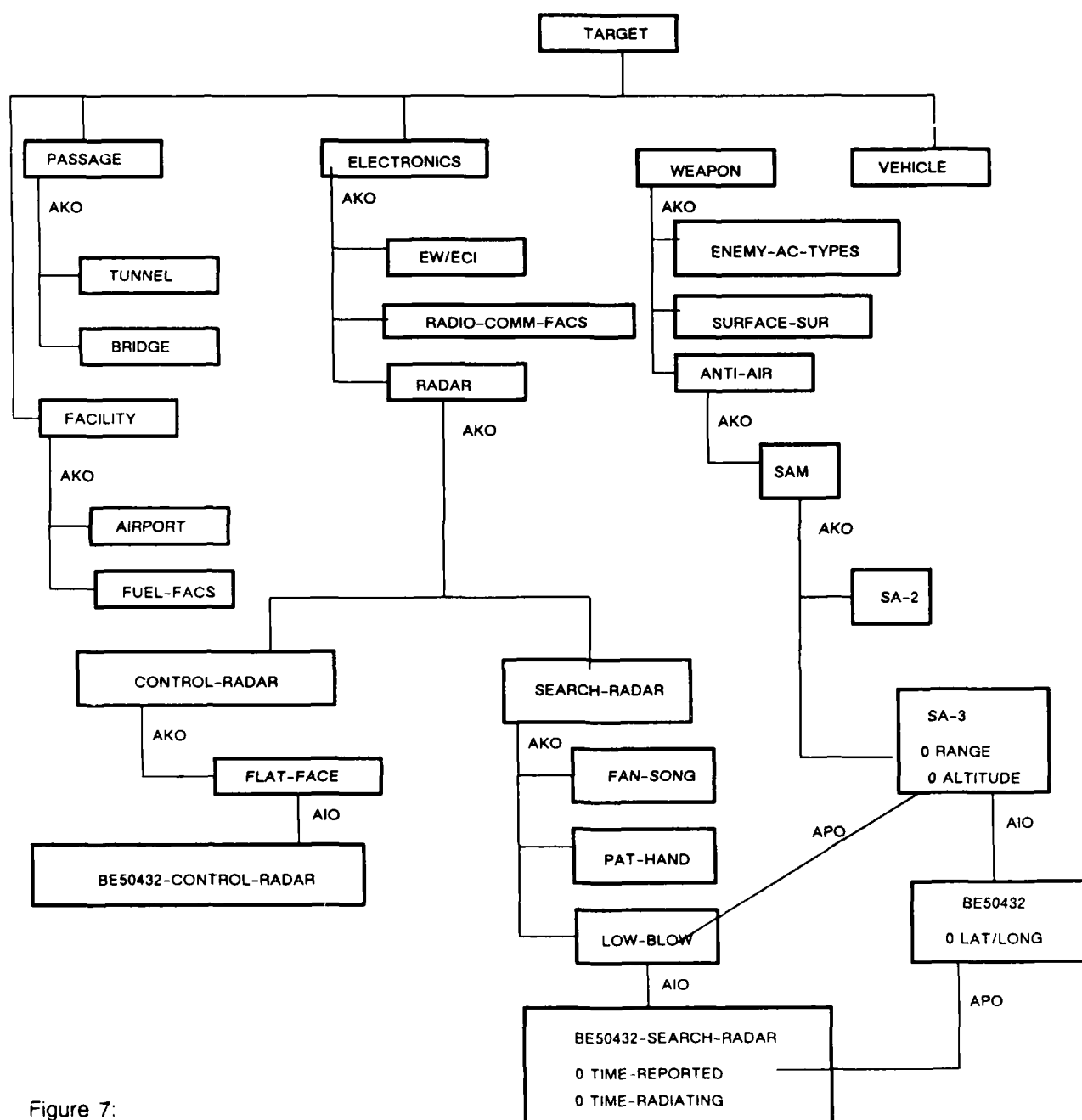


Figure 7:

A Sample of the Target Portion of the Data Base. Weather information for targets is kept in a separate structure organized by a grid of Lat./Long. values

seven of them are compared in [STEF79]. One of these was FRL (for "Frame Representation Language"), which had been developed at MIT by Roberts and Goldstein [ROGO77]. It was felt to be the smallest, simplest, and most compatible of the ones available with the desired features. FRL was written in MacLisp, but it was practical to translate it into Interlisp for KNOBS [ERIC79].

An FRL frame is a nested association list. The frame has up to five levels of embedding, indicated below.

```
(frame
  (slot (facet (datum (label message ... ))
                (datum ... ) ... )
        (facet ... ) ... )
    ... )
  (slot ... )
  ... )
```

What we have been calling an "attribute" is called a "slot" in FRL. The value of a slot is a datum under a \$VALUE facet. There are a few other standard facets which have significance to functions provided in the FRL package: \$IF-ADDED, \$IF-NEEDED, \$DEFAULT, and some others. These will be discussed below. It is possible to put arbitrary facets in a frame, and the standard facets need not all be present.

Nested substructures, such as the (LABEL MESSAGE ...) list after a datum, are optional. Only rarely does KNOBS use labels or messages, but one case shows up in the example below, the AIRCRAFT frame.

```
(AIRCRAFT
  (AKO ($VALUE (RESOURCE)))
  (AEO ($VALUE (UNIT-AIRCRAFT)))
  (POWA ($VALUE (SCL)))
  (SCL ($DEFAULT (NONE)))
  (SPEED ($DEFAULT (100 (UNIT: KNOT))))
  (INSTANCES ($VALUE
    (FIGHTER)(TANKER) ...)))
```

The default value for speed is 100; a label and message are used to indicate the physical units in which the speed is expressed.

In the AIRCRAFT frame above, incidentally, the slot AEO stands for "An Element Of" -- the inverse is AGO for "a group of". An aircraft is an element of a group of aircraft associated with a unit; the generic frame for such groups is UNIT-AIRCRAFT. POWA stands for "Parts Of Which Are"; it is the inverse of an APO link. SCL is "Standard Configuration Load" or ordnance.

Notice how the domain specific slots (SCL, SPEED, etc.) are mixed together with slots that ought to be common accross domains. One can argue either side of the question "should representation of knowledge distinguish between

architectural support common to all domains and domain specific kinds of knowledge?"

FRL Functions

FRL provides a collection of functions for creating (DEFFRAME), adding slots to (FASSERT), destroying (FDESTROY), removing slots from (FERASE), updating (FPUT, FREMOVE, FREPLACE), and retrieving information from frames (FGET). The ones mentioned are just representative - the FRL manual lists sixty-three, and it was found helpful to add some new variations for KNOBS. On the other hand, KNOBS uses relatively few of the sixty-three available.

The basic FRL function for retrieving information from a frame is FGET. FGET is called with an access path (the details are in the FRL manual) to the part of the frame desired, and returns with a list of the values found.

Inheritance

If FGET is asked for the value under the \$VALUE facet of a slot and there is no value, or the slot itself is not present in the frame, it does not give up. Instead, it looks for a generic frame of which this frame is an instance, listed under the AKO slot, and gets the value from there. As many AKO links will be traversed as necessary until a value is found or no AKO link exists. Thus a value under any slot can be inherited from a higher-level generic frame.

The KNOBS version of FGET, eventually named MGET, also recognizes a single AIO link. If some rule needed to check, for example, what the THREAT-TYPE of target BE50432 was, the system would execute MGET(BE50432 THREAT-TYPE \$VALUE). Now, BE50432 does not have a THREAT-TYPE slot, but it is AIO (an instance of) SA-3, which is AKO (a kind of) SAM. And the SAM frame has a THREAT-TYPE with value SAM-THREAT.

If the value is not present and there is no AKO link to follow, MGET looks for a \$DEFAULT facet and takes the value from there.

Procedural Attachment

Besides values and frame names, the data in slots may also be function calls, and these calls may be evaluated automatically as a result of FRL actions. \$IF-ADDED and \$IF-REMOVED expressions are evaluated when a datum is added (by FPUT, for example) or removed (by FREMOVE) from the \$VALUE slot.

An \$IF-NEEDED facet is the conventional location for an expression intended to be evaluated when MGET fails to find any value, even after following AKO's and looking for a \$DEFAULT. MGET does not call the \$IF-NEEDED expression unless explicitly instructed to do so.

One might ask why KNOBS needed to use both a rule system and FRL's \$IF-NEEDED and \$IF-ADDED mechanisms. The developers thought that rule-based inference was much more expressive and modifiable than FRL's

mechanisms. The KNOBS constraint checking mechanism can be viewed as an additional gigantic \$IF-ADDED attached to various slots. Nonetheless, under peculiar circumstances (i.e., an airbase going "down"), the \$IF-ADDED mechanisms triggered forward-chaining. Under still other circumstances, an \$IF-NEEDED triggered back-chaining -- an activity usually associated with the rule system. The non-uniformity of representations among seemingly similar kinds of information -- and the relative lack of guidance from the KNOBS architecture -- are present because these issues were not resolved by the KNOBS experiments.

THE FRL DECISION

Several decisions concerning FRL and the fact base combine and interlock to such a degree that they cannot really be discussed separately. The decisions were:

- * To use a large fact base (700+ targets, etc.)
- * To use the "frame" to organize this database
- * To use the FRL implementation of "frames"
- * The decision to use a DEC PDP-10 running Interlisp.

The decision to have a relatively large fact base was one of the wisest decisions made in those early years. This richness is one of the features that most clearly separates KNOBS from other "toy" knowledge-based systems. A heavy price was paid, however.

Remember that a PDP-10 computer uses an 18 bit address to point to a 36 bit quantity. The address space is thus about equivalent to today's personal computers (e.g., IBM 8086-based PCs have a 20 bit address space to bytes). This address space is simply not large enough to contain a sophisticated AI system like KNOBS. The AI researchers of that era either concentrated on small examples, or went through much pain and agony to write some sort of "manual memory management" to force-fit their larger applications into the small machine. The KNOBS research team spent a man-year building, among other things, the "frame swapper" trying to live in a small address space.

As we will see, KNOBS used the rule and constraint-checking systems to do most of its inference. For the most part, frame-based inheritance and the "daemons" (e.g., \$IF-ADDED and \$IF-NEEDED procedures) were seldom used. Even when used, there were alternatives available. This combined with a continuing search for performance to drive the developers to "short circuit" much of the automatic inference supported by FRL. Indeed, only a small part (three or four pages of source code) of FRL is actually used. The result is a fast system, but one cannot now put an \$IF-ADDED on a slot with any confidence that the daemon will be inherited or that the particular values being watched for by the daemon will actually be put into the slot in the "standard" FRL way.

Remember the original goal for using FRL: to organize the fact base around a particular set of objects, with each object being represented as a frame. This organization was needed to avoid expensive linear searches of the entire fact base to find the value of some property of something: no search was needed to find the atom to which the corresponding frame was attached. However, each slot access to that frame still uses a linear search of the names of the slots. Similarly, a linear search is used to find facets within the slots. If the slot access tries to inherit, then a further linear search of the class hierarchy also occurs. These searches are NOT inherent in "frames," but result from the particular implementation of the notion of "frames" in FRL. More recent notions of *inheritance* found in *Flavors* do all the inheritance at compile time, so inheritance has no particular time-cost performance impact.

With the clear vision of hindsight, KNOBS might have been better off to have implemented its own version of "frames" that was perhaps more complex but more computationally efficient (in particular, one that forward-chained inheritance and compiled daemons). Presumably, if the frame-style inference techniques (i.e., daemons) had not extracted a price even when they were not used, then they would not have been short-circuited.

One of the interesting theoretical contributions of KNOBS was its exploitation of multi-directional inheritance. This facility was necessary because the bare implementation of FRL did not distinguish between set membership (AIO -- "an instance of") and subset (AKO -- "a kind of"). Correcting that elementary logical oversight involved making inheritance trace multiple slots. But once that step had been taken, the obvious generalization of selecting sets of slots along which to inherit was taken as well. Specifically, the APO (a part of) relation was inherited through when asking about the location and radiative characteristics of something.

To summarize, the KNOBS effort revealed that inheritance in a homogeneous tree hierarchy as supported by FRL was not sufficiently general for its application. The semantically necessary addition of the AIO link provided the opportunity to generalize the notion of inheritance to multiple paths specified by sets of slot names. Simultaneously, the implementors began to feel that inheritance was too costly to be used all of the time, especially if the developer "knew" (but see below) that inheritance would not find anything useful.

The paragraph above should be taken as a caution: there are many cases when the developer "knew" that inheritance wouldn't find the relevant information, so inheritance (or activation of \$IF-NEEDED, or a call to the rule-based inference mechanism, or what have you) was short-circuited. It became the responsibility of later developers to examine each mechanism for every new fact to make sure that some previous developer hadn't added an efficiency hack that now prevents the new fact from being used. The developing KNOBS architecture may have discouraged such "hacking," but the implementation did not enforce any standard utilization of inference mechanisms, and the development environment extracted a heavy penalty for doing any "unnecessary" computation.

THE EFFECT OF FRAMES ON INFERENCE

When the list of facts was replaced by the frame system, factual clauses had to be treated differently. In the present KNOBS system, a general factual clause has the form:

(slot frame value);

that is, the relation of the clause became the name of a slot in the frame appearing as the first argument, and the second argument is the value for that slot.

If the slot, frame, and value are all constants, INFER checks to see that the given value is in fact the value for the given slot. If the frame has been bound to a constant, and the value is a variable, INFER will bind the variable to the value for the slot. If the frame is still a variable, however, INFER will fail, since it is impractical to search the entire data base for a frame that has the needed slot.

In KNOBS, the slot is always a constant. As a general observation, it is hard to see why any rule in the KNOBS application domain would need to "find some arbitrary relation between X and Y." This observation is not valid in other applications, for example, in mathematical problem solving one might ask to find a functional relationship between X and Y such that $X = F(Y)$, such that F has an inverse (or is associative, or commutative, etc.).

What capabilities, if any, were given up by INFER failing to try to bind the variables in the following situations:

(slot=VARIABLE ...)

(slot=CONSTANT frame=VARIABLE value=CONSTANT)?

The first form generally doesn't arise; it is saying something like "What is the relation between ..." or "For some relation between ..." The second form is generally useful; it is saying something like "What are the Xs with something filling some role?" We'll see later that KNOBS had a mechanism for enumerating candidates to fill some slot of a frame; one can only speculate what might have happened if KNOBS had included an efficient way to use those generators in the rule-based system.

The Format of the Rule Evolved

Standardizing the format of a clause to (slot frame value) is the third change in rule format that had occurred in the evolution of KNOBS. Recall that originally the first clause had an English-like representation. Then the format of the clauses were normalized to resemble LISP's so-called Cambridge-Polish notation with relation first followed by arguments. Now the form of a clause in the rule's hypothesis has changed yet again.

Constructing a "good" rule continued to involve some engineering. For example, the new INFER would not search for the first argument of a fact. This implies that the order of hypothesis in a rule can be significant. When INFER

finds the value of a variable in a hypothesis, it substitutes that value for occurrences of that variable in later hypotheses. Thus, if a rule has the hypothesis:

(RADAR BE50432 ?X)
(TIME-REPORTED ?X ?T)

INFER will substitute BE50432-SEARCH-RADAR for ?X, substitute it into the next hypothesis, and find ?T with no trouble. If the TIME-REPORTED clause came first, however, INFER would fail because the first argument, ?X, would still be unbound.

Multiple Values

It is possible for a frame to have multiple values in a single slot (when there is more than one datum in the list under the \$VALUE facet). In this case INFER must try the values in the slot one at a time until the "match" succeeds or INFER runs out of values. That is, the interpretation of a predicate on a multiple-valued slot is "there exists" rather than "for all."

This possibility was an accident of notation, not a carefully constructed part of "frame" theory. The implementors of KNOBS noticed the opportunity and took it. In retrospect, this is a good example of how a trick of notation prevented an advance in KNOBS's theory of planning. Semantically, it would have been clearer to have said that some slots are "set valued" with a system-wide convention that LISP lists without duplicated elements are the standard representation of a set. Then it would have been simple to talk about a separate \$SET-SIZE facet of a slot that contained the cardinality of the set, etc. Had this alternative course been taken, it would have been very natural to talk about constraints on sets as well as constraints on the elements of a set. Unfortunately, because the notation for multiple values was so convenient, the easier, less profound, course was taken.

Although the two notions "set-valued" and "multiple-valued" appear to be the same, a little contemplation will reveal that one cannot write certain rules or constraints about a multiple valued slot. Examples of desirable rules and constraints that refer to *sets* are shown below.

The package should contain at least one
jamming mission.

There should be no more than two destroyers
in a "fly the flag" mission.

Every aircraft in the package should have
IR-guided weapons.

The KNOBS implementors might say that the failure to see this was in part due to the peculiar lack of need for it in the Air Force domain. When it showed up in a re-application of the KNOBS architecture to the Navy "fly the flag" mission planning task, the problem was re-thought. However, this defense is somewhat

self-justifying: because the notion of "set-valued slots" was not introduced, it was not utilized. Because it wasn't utilized, it "obviously" wasn't needed.

We must also ask why the "there exists" interpretation was selected. Recall that in the type of application intended for the developing KNOBS architecture, the rules were typically dealing with whether an event had occurred and with recommendations for which resource to allocate on the basis of features of the mission. The typical English phrasing of these rules starts with "If there exists a ..." or "In the presence of a ...". Pragmatically, the "there exists" interpretation of a clause involving a multiple-valued slot is probably the correct interpretation in almost all cases.

Inheritance

INFER uses MGET to obtain the value in the slot; this means that it can obtain a value by inheritance from a higher level frame, rather than directly from the given frame. The special keyword IS-A is used to indicate a relationship between frames that is inherited through a sequence of AKO links preceded optionally by an AIO link. For example, the clause (IS-A BE50432 RADAR) succeeds, since BE50432 is AIO LOW-BLOW, LOW-BLOW is AKO SEARCH-RADAR, and SEARCH-RADAR is AKO RADAR.

When inheritance is used, the audit trail indicates that by using the keyword INHERITANCE instead of DATA. Thus, the audit trail for the radar example is:

(INHERITANCE (IS-A BE50432 RADAR)).

MISSION PLANNING -- THE PLANNING TASK

KNOBS was an experimental program whose goal was to demonstrate the applicability of artificial intelligence techniques to Air Force tactical mission planning.

It was felt that a very useful capability for an AI-based planner to have would be the ability to tell the human planner what it "knew" about acceptable choices before the human selected a particular item. KNOBS has several options for supplying the planner with this kind of information:

- o a list or range of possibilities;
- o a list ordered by preference (the program can explain its ordering);
- o a proposed set of consistent selections for all remaining items (in effect, automatic planning).

The original performance goal for KNOBS's planning capability was to check the human planner's decisions for consistency and correctness. However, it was clear from the beginning that a very good plan checker could serve as the basis for a semi-automated plan generator due to a peculiar feature of the application: all missions in any category (e.g., Offensive Counter Air, Fighter Escort, Refueling) closely fit a stereotypical mission profile.

Stereotypical Planning

Mission planning is accomplished by filling in a number of blanks on a form. Furthermore, the blanks in the form are not filled with cross-references to other plans, but with entities from the data base. The same is not true of more general forms of planning, e.g., planning a sequence of motions of a robot manipulator to assemble a model airplane engine. We use the term "stereotypical planning" to refer to this special type of planning problem.

One of the critical observations made by the developers of KNOBS was that stereotypical planning problems could be solved relatively easily by a generate-and-test paradigm. That is, since all the potential slot fillers were already known, were relatively few in number (certainly fewer than 100 in all cases) and in the data base, they could be easily enumerated (or computed, as is the case with transponder frequency). For the few slots that could not be enumerated (for example, time of departure), a simple iterative scheme was used; when it broke down, so did KNOBS. In these circumstances, if the plan checker were good enough, the incorrect slot fillers could then be rejected. If candidates for a particular slot were exhausted, then the planner could back up to a previous slot and try another possibility.

Note that the generate-and-test paradigm does not in general provide a computationally tractable solution to planning problems. The surprising result of the KNOBS investigation was that it does solve stereotypical planning problems. The original KNOBS scenario diverged from the strict stereotypical

planning problem in only one respect: an OCA mission may need to be refueled by another mission in order to make the round trip to the target. This refueling mission might not be (and, in general, would not be) in the data base. The OCA mission thus has a slot that must be filled by something that, in turn, must be planned. Planning refueling was not completely supported in the final KNOBS scenario.

Multiple-frame Planning

It is tempting to characterize stereotypical planning as being "planning in which only a single frame's slots are being filled at a time, and all frames are independent". It appears to be the case that planning applications stretch along a continuum with strictly stereotypical planning at one end and planning for multiple interacting and communicating agents at the other. The classical planning problems for one-armed robots and mobile robots appear to be somewhere between those two extremes.

During the course of the KNOBS experiments, multiple-frame plans were produced. The example usually used involved refueling. Suppose each OCA mission requiring refueling had its own refueling mission assigned to it, so that the correspondence between OCA and refueling was one-to-one. Although those two missions were represented as two frames, each filling some slot of the other's, it would have been equally reasonable to have built a single "coordinated OCA/refueling" frame to hold both plans, provided the supposition holds.

In theory, OCA missions may be refueled several times by different refueling missions during its flight. Similarly, a single tanker can refuel several OCA missions. These multi-frame planning problems usually cannot be viewed as alternative representations of a single "coordinating" frame. We must then ask how well the stereotypical planning techniques extend to these slightly more ambitious planning problems. The remainder of this chapter attempts to answer that question.

Resource Conflicts are not Stereotypical

Let's assume for the moment that all planning problems can be viewed as filling in lots of slots in multiple frames, where the potential fillers for slots (which may be "set valued") are either elements of the data base or other frames. If the planning problem is not stereotypical, the computer system will have a very difficult time gathering together all the restrictions and inter-relationships that must be maintained among the various slots in multiple frames. Indeed, consider the special case of planning when resources are scarce. Competition for a scarce resource can be detected, but cannot in general be predicted. Neither can competition for a scarce resource be explained in terms of which slots have contradictory values: if 50 missions compete for a resource that can only satisfy 49, then strictly speaking all 50 missions are "wrong" in the sense that they are all involved in a conflict.

The KNOBS scenario was constructed so that resources were either present or absent, but almost never scarce. That is to say, if Hahn had F-111Es, then the data base would report that Hahn had lots of them. In the course of a normal

demonstration, where only a dozen or so missions were being planned, there was only one example case in which some resource became scarce. In this example, the conflict was trivially resolved: the wing with the scarce planes was simply not selected as the wing to supply the aircraft.

The paragraph above illustrates a subtle distinction: it is not the scarcity of resources, but the competition over resources, that tends to drive the application away from being characterized as "stereotypical." Indeed, stereotypical planning could be characterized as planning when all subgoal interactions can be characterized and resolved *a priori*. Conflict over a scarce resource is a simple example of subgoal conflict.

"Set Valued Slots" and Stereotypical Planning

The principle paradigm for solving a stereotypical planning problem is that of "generate and test." The core of KNOBS technology revolves around using rules and constraints to be clever in generating possibilities, ordering those possibilities, and testing them. Any property of an application that makes possibilities difficult to enumerate contra-indicates this paradigm.

Such a characteristic of a planning application that tends to drive it from the "stereotypical planning" regime is the presence of set-valued (or multiple-valued) slots. There is no particular problem in filling a set-valued slot if the frames in such a slot have a single-valued slot pointing back (that is, there is little difficulty if the multiple-valued slot simply holds the inverse of a many-to-one map). The difficulty arises when one attempts to generate potential sets of possibilities that are themselves sets. Assuming one can easily generate the potential members of the set, the number of potential sets is the powerset (of size 2^N) of the N potential members.

Mixed Initiative

A critical decision was made: KNOBS should support a human planner in a "mixed initiative" mode. That is, the human could make what ever selections he wished (including incorrect or contradictory selections). KNOBS could complete a consistent partial plan using the generate-and-test approach outlined above. In retrospect, this was the right decision: it also is one of the more distinctive features of the KNOBS system; most AI planning systems "want" to make as many decisions as they can, with only the remaining decisions left to the human. In other words, most AI planning systems took all the initiative, and relinquished it only when they failed (incidentally leaving such a mess that the human could only start out again from scratch). In contrast, conventional (non-AI) approaches had equally rigid ideas about initiative, with most of the initiative usually forced onto the user.

Support of a "mixed initiative" and the application being one of stereotypical planning may be linked together. Suppose resources are fairly scarce, and there is much conflict over which plan element gets to use which resource. Under these circumstances, the "mixed initiative" can easily degenerate into a conversation where the user says "Do use these things here, and solve the rest your self" and the computer responds "I cannot complete the problem because EVERYTHING IS WRONG" because all completions of one of the user's

requests involve a resource conflict implicating all other user requests. This observation leads to the speculation that "mixed initiative" is practical only if the planning problem is stereotypical.

If the planning problem is not stereotypical, then the communication between man and machine must center not on the solution, but *the solution strategy* that should be used. Only when the machine "understands" the user's problem-solving strategy can mixed initiative become practical.

In order to support a mixed initiative, KNOBS must be able to take the initiative -- KNOBS needed to be able to "autoplan." In order to "autoplan" effectively, KNOBS needed to be able to establish an a priori ordering of potential slot fillers. All potential slot fillers are either in the data base or generated by "dumb" increments of a simple integer (e.g., the number of aircraft or the time of departure). The key to being able to autoplan is being able to correctly reject unacceptable slot values as early as possible. This is why the emphasis in KNOBS rapidly turned to effectively using multiple criteria to reject "bad" choices for filling slots, and to flexibly using ordering criteria to sort through alternatives.

From MICROKNOBS to KNOBS

MICROKNOBS originally recommended munitions on the basis of target characteristics (albeit using a few simple rules). KNOBS eventually expanded this service considerably, to encompass the choice of all the parameters in a tactical air mission.

As a historical side-light, KNOBS -- unlike its predecessor MICROKNOBS -- was unable to order ordinances for most of its history. It is not uncommon for a new implementation to be far weaker than its predecessor. One should not become concerned about "lost capabilities" early in a reimplementation effort.

An offensive counter-air mission plan, for example, begins with a choice of target and continues with airbase, aircraft, etc. It starts with a blank form something like this:

The diagram shows a rectangular frame with a brick-like border. Inside, there are three horizontal rectangular boxes stacked vertically. Each box contains a label followed by an empty oval for input. The labels are 'Aircraft:', 'Airbase:', and 'Target:' from top to bottom. Below these boxes, the text 'OCA Plan' is centered.

Figure 8: Frames as Plans

Targets were always selected manually. Airbases, aircraft (types), etc. were assumed to be pre-selected by the user before the auto-planning by KNOBS began, and (of course) all candidates were in the initial data base.

With the change to using frames in KNOBS, the quantities above became slots, with the choice being placed added as the value of the slot. Constraints values selected for the plan are thus intimately concerned with the slots of frames.

MICROKNOBS did not include constraint checking per se. Rather, it used a number of "recommendation rules" (as described earlier) to suggest appropriate ordinance.

KNOBS USES CONSTRAINTS

Simple and Complex Constraints

The notion of a constraint is central to planning. In stereotypical planning, the choices for items to fill slots are constrained by the relationships that must be maintained among them. For example, the designated aircraft must be available in sufficient quantity at the airbase and have sufficient range to make it to the target and back. Similarly, the aircraft must be able to carry the proposed ordinance, which in turn must be appropriate for the target. A planning consultant should have sufficient knowledge to check these constraints and automatically warn the planner of problems.

A "simple" constraint is a relation among slots within a single frame. As mentioned earlier, it may be necessary to create two or more mission plans that

are interrelated. Interrelated missions involve constraints that span two or more related missions; these are called complex constraints.

The Generic Mission Frame and Template

Different types of missions have different sets of slots. One might expect that the proper set of slots for a newly created mission frame could be determined by looking at its generic frame. There are conceptual reasons for not organizing the implementation that way (see following discussion). The original KNOBS developers thought that putting the mission parameter slots into a separate frame called a "template" made the system more readable; it turns out that subsequent generations of developers believe "templates" are difficult to read and write.

The real reason for using "templates" stems from a desire to separate "working slots" from "planning slots." Typically, a frame for an OCA mission contains a number of slots maintained for "internal" reasons, e.g., slots to hold various "audit trails" returned from the rule interpreter. Similarly, the "generic" OCA frame contains a number of auxiliary slots that contain annotation of various kinds (for example, AIO "an instance of"). The KNOBS researchers argued that the essential planning information should be placed into a separate "template" that contains only the slots that need to be filled in to specify the plan, and the constraints on those slots that must be satisfied if the plan is to be free of conflicts.

Besides the "template," the generic mission frame of each type of mission has a slot that lists all the individual mission frames of its type; so a display or printout of the generic frame for maintenance purposes would be too unwieldy if it included all the mission slots as well.

Currently, the organization of mission frames looks roughly as shown in Figure 9. The relationship of a template to a generic mission type frame is ATO (A Template Of).

CONSTRAINTS

Since MICROKNOBS had a rule interpreter (and a number of rules related to planning OCA missions), the developers of KNOBS naturally tried to use that machinery to check constraints. They also wanted the explanatory and modificational benefits afforded by rules. However, a round-about method was employed: constraints are represented as functions. Some of these functions invoke INFER to utilize the rule interpreter.

By the end of KNOBS's evolution, the general format for a constraint within a template had become a LISP function (of N arguments), and a list of slot references (N of them), plus some auxiliary information. The usual representation for a LISP function is simply its name. A constraint looks like this:

```
(constraint-function (slot-reference1 ...
slot-referenceN) timely-specification flags).
```

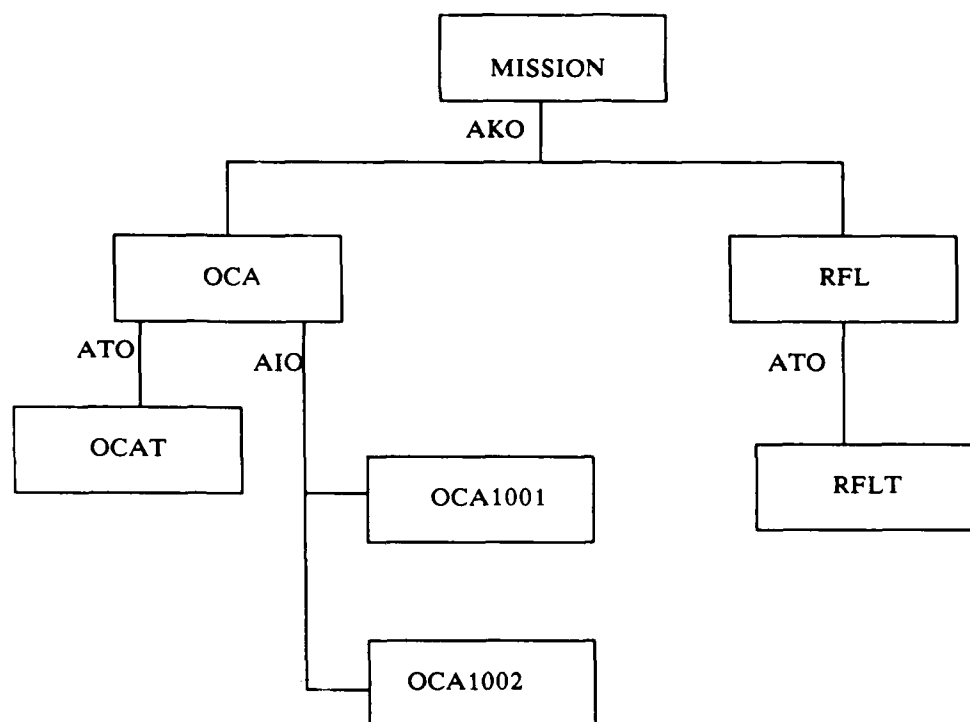


Figure 9: Mission Frame Organization

Each slot reference is either the name of a slot (in the "current" mission's frame) or an access path giving a trail of slot names to follow from the current frame to a "remote" slot.

The organization above was not "designed;" rather, it evolved through experimentation and reconsideration. The first implementation simply had a function name and "local" slots. The "timely-specification" (see explanation below) was added later, as were the various flags. Finally, the "remote slot" designation was added. Subsequent to all of that evolution, multiple-valued slots were added. Unfortunately, multiple-valued slots raise a "there exists" or "for all" interpretation ambiguity: should the constraint be satisfied for all the frames' slots, or is it sufficient to satisfy just one frame's slot? Midway in KNOBS evolution, the implementors decided that the constraint code should decide on which interpretation to use. At the end of the KNOBS development, multiple-valued slots were not really supported when they were part of a remote slot path.

One can only speculate why the KNOBS designers chose to implement a "remote slot" as a list of slot names associated with a constraint, rather than in some more uniform manner by attaching a map of "remote slot names" to "paths" to the generic OCA mission frame (or some other appropriate frame).

This alternative is certainly more concise and more flexible (the route could then be computed incrementally, and if the organization of frames was changed, it would only need be recorded in a single place, rather than requiring every constraint to be rephrased). However, given an implemented constraint checker, it was probably implementationally easier and safer to make the code dispatch on the type (atom vs. list) of the slot reference than to invoke a more complex mapping facility on all slot references. This is a prime example of the danger of allowing an implementation to dictate conceptual organization.

Non-Rule Constraints

While many constraints are based on inference rules, the KNOBS constraint-checking architecture made it possible to have constraint functions that perform other computations, instead of calling INFER. One can argue that it would be inappropriate to express some constraints as rules, because they involve special algorithms.

It should be admitted that almost any computation can be embedded in a rule as a side-condition. However, the rule would be nothing more than a decorative envelope for the computation. It was felt to be inelegant and inefficient to invoke the inference mechanism unnecessarily when all that was needed was to call a certain computational function.

Non-rule constraints can be written in such a way that some of their arguments are optional, or certain combinations of arguments can be used in place of others. These considerations led to the design of a constraint timeliness "calculus" for representing the needed arguments. These notations will be described in the following section.

Rule-Based Constraints

In the early days of KNOBS, it was believed that most constraints would simply call INFER to calculate whether there was a conflict between a proposed slot value and other considerations. This view of constraint-checking called for a slightly different view of how rules were used. Instead of generating a recommendation, a rule would now generate a warning that some mission parameters were in conflict with one another. A typical rule with a conflict conclusion is this one:

```
IF:
  1: THE TARGET OF A ?MSN RADIATES
AND 2: THE AIRCRAFT OF THE ?MSN IS NOT AN F-4G

THEN:
  1: THERE IS A SERIOUS CONFLICT BETWEEN THE TARGET
AND THE AIRCRAFT OF THE ?MSN.
```

It is important to give the user warning of conflicts as soon as possible, that is, as soon as all the relevant slot values had been entered, or when one was changed.

At this point the constraint is said to be "timely." For each slot in a mission template, there is a list of constraints indicating which conflict rules might be

triggered by a new value for that slot. The constraint lists which other slots had to have values in order to check the hypotheses of the rule.

For example, the above rule is invoked by a constraint:

(TGTACP1 (TARGET AIRCRAFT)).

This constraint is associated with both the TARGET slot and the AIRCRAFT slot. When either of these has a value entered into it, KNOBS, seeing this constraint, will check whether the other slot also has a value, and, if so, will indirectly invoke the rule. "TGTACP1" is not the name of the rule, but it is the name of a function that calls INFER with the conclusion of the rule.

Daemons

A daemon is a non-rule constraint that never fails; it is executed for effect. Generally, it was often used as a mechanism for maintaining consistency between data controlled by the user and other data not under the direct control of the user. This can be thought of as a response to a need for a data-driven architecture (although there were \$IF-ADDED and \$IF-REMOVED facets already implemented in FRL). The notion of using constraint firing and the timeliness calculation was so attractive that the developers wanted to try using that single mechanism for other purposes.

As mentioned in the description of the availability constraint, a daemon is used to update aircraft allocation records when the relevant mission slots have been filled. The availability daemon has the same arguments as the availability constraint, except that none of its arguments is optional. The daemon has a lower priority than the constraint to ensure that availability is checked before the allocation is recorded.

Daemons are also used to fill in certain mission slots that are supplied automatically by KNOBS when possible. In an OCA mission, for example, there is a CALL-SIGN slot that can be assigned as soon as the UNIT and ACNUMBER slots are defined and unchallenged.

UnDaemons

Along with daemons, KNOBS used a similar mechanism - undaemons - to automatically retract decisions. Much difficulty was encountered throughout the project in making sure that the undaemons could fire at the appropriate time. One suspects that much of the complexity of the timeliness calculus was needed for controlling when an undaemon could fire.

Retrospective Review of Constraints

Many of the early expectations concerning constraints did not come to pass. Relatively few of the constraints used the rule interpreter in non-trivial ways. Instead, most of the constraints were directly coded into LISP. Similarly, most of the rule predicates also evaluated LISP code directly while seldom recursively invoking the rule interpreter.

There are two problems with allowing arbitrary LISP code to be used in an AI architecture. The first problem is one of capabilities: the architecture cannot (in general) perform any manipulations on the LISP code. It cannot explain the code to the user (beyond "canned" explanation which may or may not correspond to the actual actions of the code), it cannot adapt the knowledge embedded in the code to any purpose other than what the programmer intended, and it cannot "see" what the code does unless the programmer bypasses the architecture and uses inference engine primitives to update all the appropriate internal memory structures. The second problem is philosophical: insofar as a theory of planning says "at this point, arbitrary things happen at the programmer's whim," the theory does not explain how planning is performed. KNOBS is an extreme case: the "theory" in KNOBS does not explain how constraints should be checked (the constraints are LISP code, so anything can happen), nor does it explain what rules are (side conditions invoke arbitrary LISP code, so anything can be done).

Some of this preference for using LISP code resulted from concern with the time-response of KNOBS in demonstrations. Constraints written in LISP were much faster. Adding a constraint that uses rules typically involved much more effort on the part of the implementor than adding one that used LISP directly: new rules had to be written, tests for type hypothesis clauses of the new rule had to be added, etc. Finally, the rule-interpreter was under continued development, and so was "flaky" much of the time; self-preservation encouraged the implementors to use LISP rather than rules.

In defense of KNOBS, we should be aware of a general feeling within the community at the time that one should first get a rule-based system working, and then put the knowledge into rules. This was the approach taken in KNOBS: it resulted in an implementation in which most knowledge was in fact recorded in LISP code.

Because there were so few rules, nobody tried to make it easy to install new ones. Because it was difficult to install rules, constraints were tested using logic expressed in LISP "just to get them working." It was always their intention to translate most of the coded functions into rules, but such a translation never took place.

CONSTRAINT TIMELINESS AND PRIORITY

There are two reasons why constraints might be prioritized: one reason is for efficiency, the other is for more responsive explanations to the user. In the KNOBS architecture, the latter reason is dominant, although the developer's original motivation was an even mix. A constraint's priority is a measure of how useful that constraint's failure is to the human user. Similarly, a constraint is "timely" when the user would be told of the constraint's failure should such a failure occur.

On the "efficiency" side, there is an intuitive argument that constraints simply should not be tested when the situation renders them irrelevant or nonsensical: don't worry if the thing in the AIRBASE slot has the right aircraft until you know that the thing is in fact an airbase. In hindsight, there are probably

conceptual categories for constraints having to do with the kind of consequence a constraint has (e.g., the mission plan is *nonsense* if the AIRCRAFT is not an aircraft, but would only fail if the aircraft were inappropriate for the target). KNOBS tried to capture both of these dimensions in a single quantity called *constraint priority*.

Constraint Priorities

When a new value is proposed for some mission slot, there may be more than one reason why it should be rejected. For example, suppose a mission has already had the following selections made for it:

TARGET: BE70501-RUNWAY
AIRCRAFT: F-111E

and now Mildenhall is suggested for the AIRBASE slot. As it happens, the distance from Mildenhall to BE70501 is beyond the range of an F-111E. That problem could, perhaps, be fixed by planning a refueling mission. However, there is a more serious problem: there are no F-111E's at Mildenhall. Clearly, only the latter reason for rejecting Mildenhall should be given to the human planner.

The constraint testing existence of aircraft at an airbase is therefore given a higher priority than the range constraint. In general, the constraints relevant to each slot are listed in order of priority, so that the more important conflicts will be identified first.

Timeliness and Trapping

The fundamental test of "timeliness" is that a constraint is not "timely" if one of the slots needed by the constraint is not available. For example, a slot without a value is certainly not available. There is also an interaction between timeliness and priorities. A constraint is not considered timely if a slot it needs is involved in the violation of a higher-priority constraint. In this case we say that the slot is "trapped". The precise operation of the timeliness computation and "trapping" will be explained in the next chapter.

Notice that "timeliness" is actually serving a dual purpose: it is blocking a constraint check when some of its arguments are not available, and it is blocking a constraint according to a particular model of what and when a user wants to be told about a problem.

This interaction of priority and timeliness is dynamic, in the sense that a change in one slot may have a ripple effect: by violating one constraint, other slots become trapped, causing another failed constraint to lose its timeliness, causing it to release slots it had trapped, making another constraint timely, with perhaps further effects.

The process for sorting out the effects of slot value changes on the timeliness of constraints is illustrated in the next section, using a stripped-down version of the mission scheduling task as an example.

Aircraft Availability -- The Effects of Timeliness

Before giving a complete explanation of the timeliness computation, let's look at an example. This example will help pin down some of the concepts already presented, and establish some intuitions about what priorities and timeliness involve.

An important example of a non-rule constraint in KNOBS is the one that checks aircraft availability. It determines whether a required number of aircraft from a given unit will be available for a mission with a given schedule. The calculation uses a general algorithm suitable for handling many similar questions of resource availability.

Records of the allocations of aircraft are kept in the AVAILABILITY slot in individual unit-aircraft frames. For example, the frame 101TFW-F-111E initially has the value (17) in its AVAILABILITY slot, indicating that the unit 101TFW has an initial allocation of 17 F-111E's. When a mission plan is completed that uses some of these F-111E's, however, the AVAILABILITY value is augmented to show the withdrawal of some number of aircraft for a certain period. For example, the value:

(17 (start-time 4 OCA1001) (end-time -4 OCA1001))

indicates that OCA1001 plans to withdraw 4 aircraft at start-time and replace them at end-time.

Start-time and end-time are given in an internal time format. The start-time and end-time are calculated from the TD and TOT of the mission. The start-time is the TD minus the uploading and preparation (turn-around) time, and the end-time is the TOT plus the time required to return to the base and perform any necessary post-flight service.

The constraint AVAILP1 becomes timely when the UNIT, AIRCRAFT, and ACNUMBER are present, along with enough other items to estimate the start and end times. It checks the feasibility of the new requirement given the existing allocation record. To do this, it starts with the initial maximum allocation of aircraft and simulates the decrementing and incrementing of the number of available aircraft at the various start and end times, in order, from the existing record. If the number of available aircraft goes below the new requirement during the period requested, the request is refused.

Some time after the availability constraint is satisfied, the new allocation will be merged with the previous allocation record in the unit-aircraft AVAILABILITY slot. The allocation update is not made until all of the relevant mission items, such as TARGET, TD, and TOT, have all been entered without challenge. The update is performed by a daemon, whose operation has been explained above.

CONSTRAINT TIMELINESS AND EVALUATION

This section explores the issue of timeliness in more detail: how it is affected by constraint priorities and trapping, and how constraint arguments are expressed. The handling of complex constraints is presented, in the context of refueling mission planning. The rationale behind the notion of timeliness was presented in the preceding chapter.

TIMELINESS, TRAPPING, and PRIORITIES

The dynamic nature of constraint checking is illustrated here with a simplified version of the mission scheduling task. Suppose that the mission slots are:

DT departure time from base
AT arrival time at destination
AC aircraft type
AB airbase
UN unit
FC flight code

and that the constraints and their priorities (lower number implies higher priority) are:

Constraint	Priority Level	Domain	Meaning
TIMEORDER	1	DT, AT	departure before arrival
REACHABLE	2	AT, AC	aircraft can arrive on time
SERVICE	3	AB, AC	airbase can service aircraft
LOCATABLE	4	UN, AB	the unit is at the airbase
RIGHTCODE	5	FC, UN	the unit has the flight code

The meaning of the constraints is not really important for the example; what counts is the priority of each constraint and its domain, the set of items it checks.

A planning schema is shown graphically in Figure 10. The horizontal lines join elements of the domain of the named constraint. The vertical lines represent the transmission of values down through the different priority levels. Values that make it through to the bottom are accepted.

In the situation illustrated, all variables except DT have had values bound, but only the value for FC has been accepted (trivially, since none of its constraints are timely). The arrival time does not allow sufficient time to make the trip (the current time is considered global), so AC and AT have been trapped at level 2 by REACHABLE, as indicated by the "X"s. SERVICE was not timely, so AB

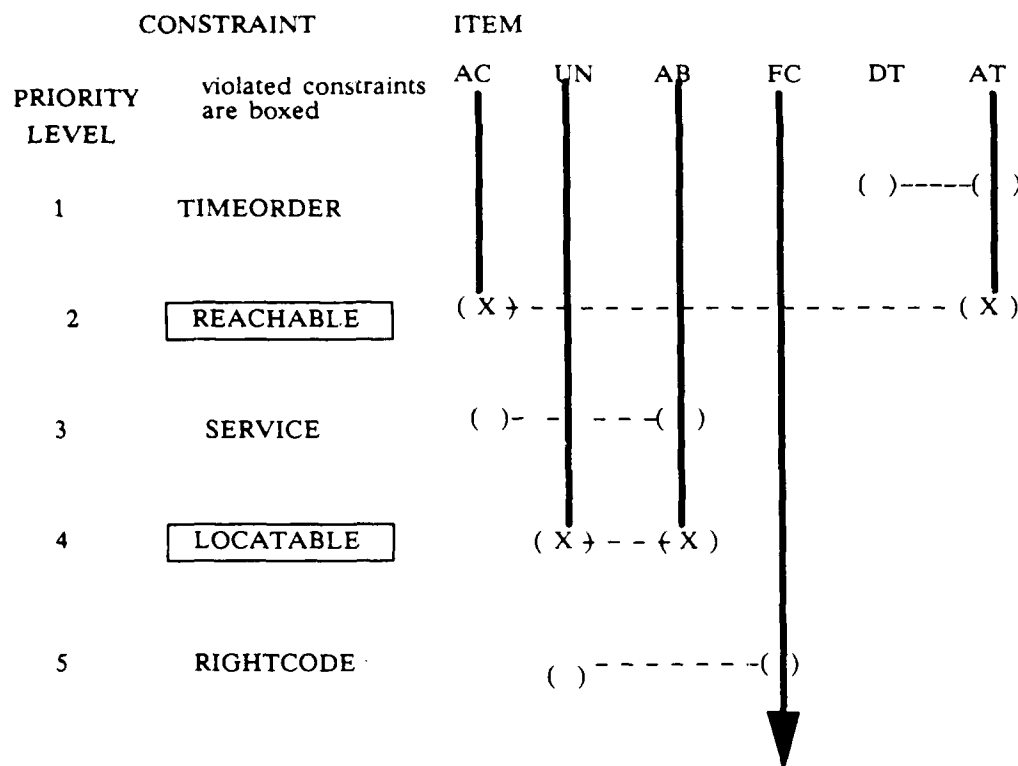


Figure 10. The Initial Snapshot, DT Still Unbound

has dropped through to priority level 4, where LOCATABLE discovers that the suggested unit is not located at the suggested airbase, so UN and AB have been trapped there. Thus far, no constraint has been passed by any variable.

Note that TIMEORDER is not timely because no value has yet been suggested for DT. On the other hand, SERVICE is not timely because of the failure of REACHABLE: we refuse to be bothered about whether the aircraft can be serviced at the airbase until resolving the allegedly more fundamental question of whether such an aircraft is able to fly such a schedule.

Suppose that the planner, upon seeing that AT is too early, decides to shift the whole flight to a more safely remote schedule and begins by assigning a new value to DT. Unfortunately, the old value of AT is earlier than the new DT. TIMEORDER, now timely, is tested and fails. This failure traps both AT and DT. The resulting situation is shown in Figure 11.

This situation is unstable, however, because REACHABLE cannot be marked as failed, much less trapping its domain, when it is not even timely. Therefore AC is released. This permits AC to fall through and make SERVICE timely. If SERVICE succeeds, then AC becomes accepted, as shown in Figure 12. If not, the reader can check that LOCATABLE will no longer be timely, releasing UN, making RIGHTCODE timely. Whatever the result of RIGHTCODE, the

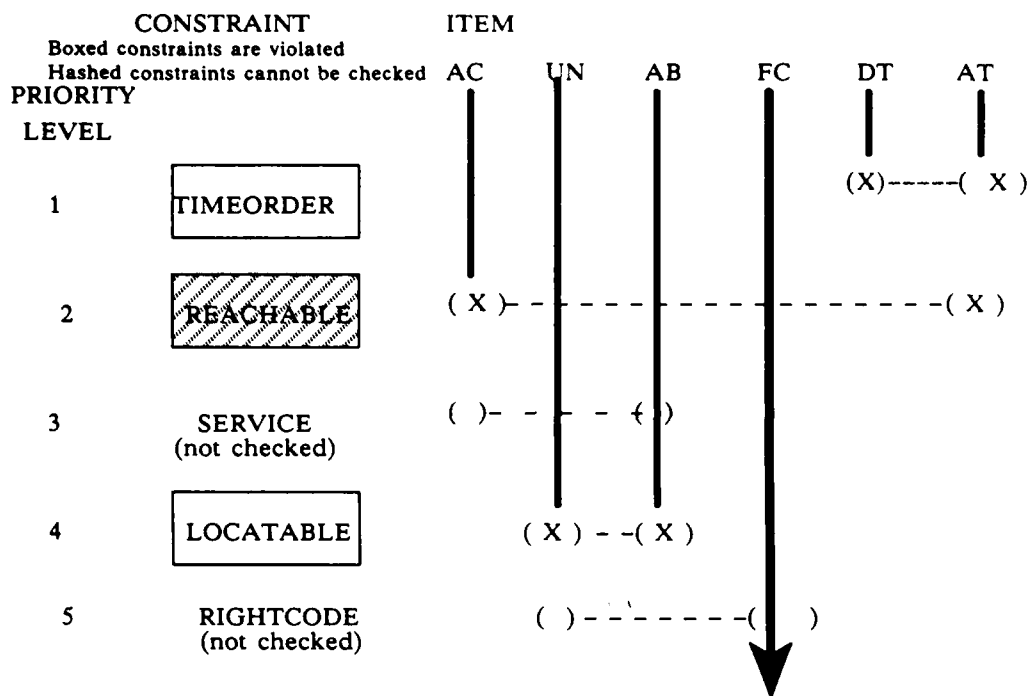


Figure 11. DT Given a Value, TIMEORDER Fails.

situation will then be stable. Notice that Figure 12 claims that AC is acceptable, even though we know it is inconsistent with the value for AT.

Re-evaluating Constraints

The checking process, as sketched, efficiently guarantees the consistency of belief in disparate plan elements, but it cannot promise that constraints will not be re-evaluated needlessly. For example, if the state in Figure 12 is altered by giving DT a new value which causes TIMEORDER to pass, then REACHABLE will be tested and fail again, and with the exact same bindings as in Figure 12. A constraint can be guaranteed unique calling sequences, of course, through maintaining full histories. However, the following heuristic is efficient and more in accord with our intuitive notion of what a planner might reasonably check.

A history list is maintained which contains only the names of variables given new values. Each constraint keeps a pointer into this history list to the item whose entry most recently caused the constraint to be timely. The constraint must also remember the result of the last evaluation (pass or fail). The structure below represents a possible history of Figure 12:

When a variable is changed, it is also inserted at the end of the history list. If a constraint becomes timely, the constraint's pointer is reset to point to the new variable at the end of the list. If any of the variables to the right of the pointer's old position are in the domain of the constraint (its set of arguments), the constraint must be re-evaluated; otherwise its old value is still good.

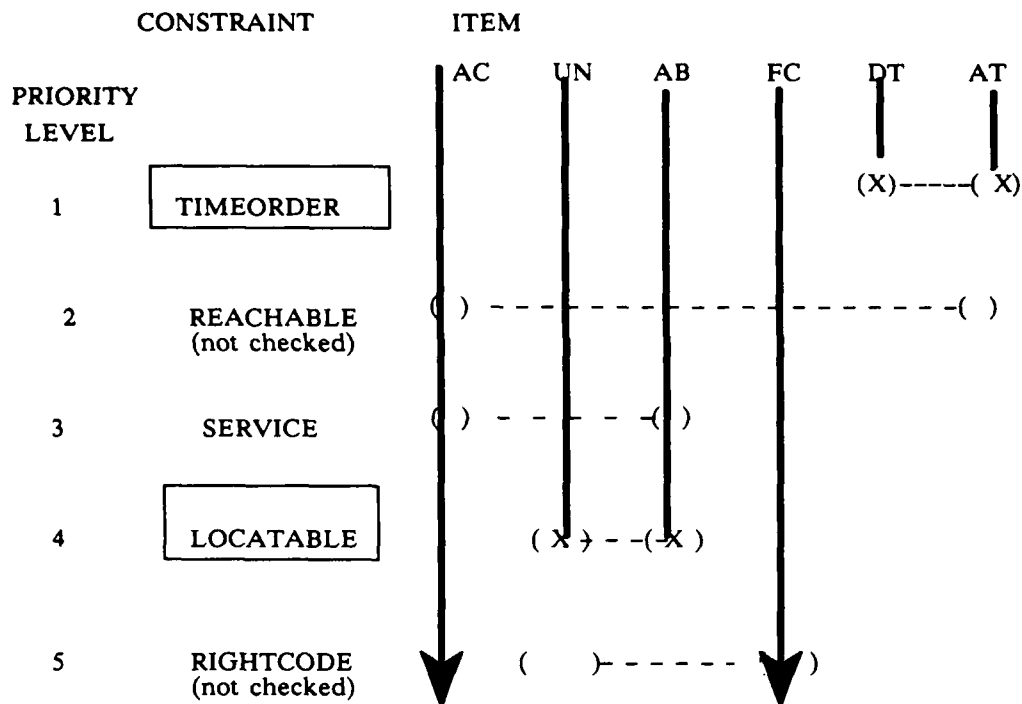


Figure 12. If SERVICE Succeeds.

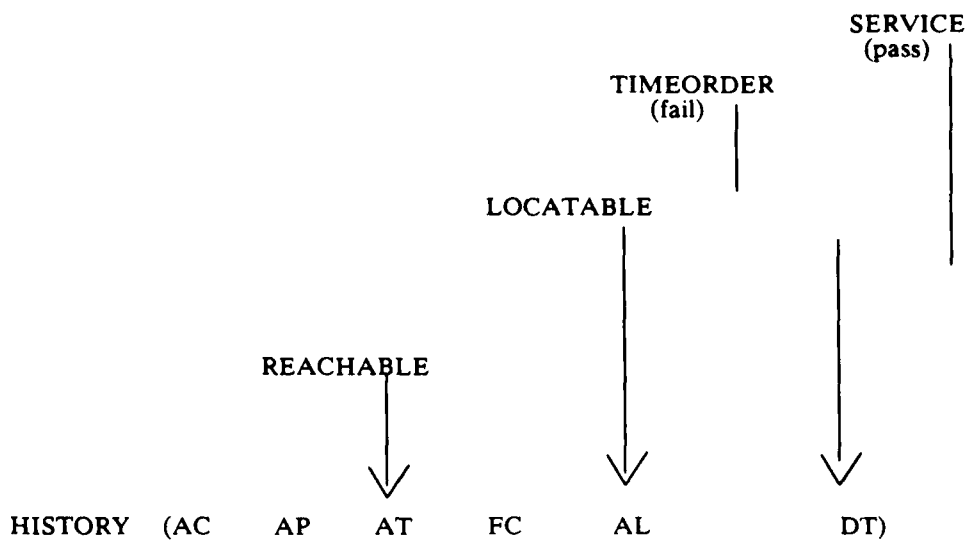


Figure 13: Possible History for Figure 12

If the history list is likely to grow very long, it can be garbage collected simply by deleting the elements prior to the first pointer.

CONSTRAINT ARGUMENTS

In general, the structure called a "constraint reference" (or sometimes just "constraint," since a constraint is simply a particular invocation of some "constraint function") has the form:

(constraint-function slot-references spec flags)

where "constraint-function" is the name of a constraint function, and "slot-references" is a list showing where to obtain the arguments. Some or all of the items in the arglist are the names of slots in the current mission frame that are involved in the constraint. However, KNOBS also handles constraints that involve values from several frames, needed, for example, to coordinate OCA and refueling missions. To obtain a slot in a frame other than the current mission, i.e., a remote slot, a list representing an access path is put in the arglist instead of just the slot name. For example, the access path

(START REFUELSVC ORBIT STORBT)

references the STORBT (start of orbit) slot in the frame named in the ORBIT slot of the frame named in the REFUELSVC slot of the current frame. This argument will be referred to by the name "START" wherever necessary in the "spec" part of the constraint reference.

The "Timeliness" Specification Language

The "spec" is a specification that indicates when the constraint is "timely"; i.e., when enough of its arguments have values available so that the constraint function can be evaluated. Note that some constraints have arguments that are optional, because it can use default values for them; also, a constraint may be evaluated with different combinations of arguments. Even when a slot has a value, it may have been involved in the failure of a higher priority constraint, in which case we say that the slot is "trapped" at a higher level. Some constraints are willing to use trapped values, others are not.

A timeliness specification is a boolean expression whose terms are slot names (or argument names, in the case of remote slots). The specification

(*AND* TARGET (*OR* UNIT AIRCRAFT))

means that the constraint can be evaluated if the TARGET slot has a value and either the UNIT or AIRCRAFT slot does. In other words, an occurrence of a slot or argument name in a specification is interpreted as TRUE if the slot has a value and the slot is not trapped, FALSE otherwise. Other boolean connectives may also be used: *NOT*, *XOR*, and *AT-LEAST* (a threshold function with argument *N*).

Although a slot is not ordinarily considered available if it is trapped, the term (*TRAP-IGNORE* slot) in place of "slot" yields TRUE if the slot has a value, whether it is trapped or not. (*TRAP-HIDE* slot) has the same meaning except that it causes the constraint to be given the value NIL in place of the actual value of the slot if the slot is trapped.

An occurrence of (*OPT* slot) in place of "slot" means that the slot is optional; the timeliness of the constraint will not be affected by whether the slot has a value, though an *OPT* term is still FALSE if the slot is trapped. Thus to specify that a constraint *always* sees a slot, use (*OPT* (*TRAP-IGNORE* slot)), which returns either the value of the slot or NIL if the slot has no value, regardless of whether the slot has been trapped or not.

Following the spec in the constraint reference is a list of flags. The only flags currently in use are *DAEMON*, indicating that the reference is actually to a daemon, and (*MASTER* ...). The use and meaning of *MASTER* is discussed below in the subsection on multiple frame co-instantiation.

Discussion of the "Timeliness" Specifications

Recall that, although constraint timeliness was somewhat concerned with making constraint checking faster (by being able to ignore constraints that are not "timely"), its primary effect was to prevent the interface from telling the user about constraint failures which were not "interesting." A third useful purpose should be recognized: since "daemons" are syntactically the same as constraints but are executed for side-effects, it is important that the side-effect not occur when the arguments to the daemon are involved in a constraint failure; the timeliness specifications allow this undesirable execution to be avoided.

For the remainder of our discussion, let's pretend that run-time efficiency is not an issue, and ask the question of whether the timeliness specification language was needed, or whether the same constraint filtering could have been accomplished automatically. That is, we will suppose all constraints associated with a frame were run when any slot in the frame was changed (complex constraints are referenced in all frames containing slots relevant to the complex constraint).

Suppose that any constraint whose arguments include any slot for which a higher priority constraint failed are elided from the list given to the user. Generally speaking, the number of constraint failures told to the user would be fewer in this scheme than in the timeliness scheme. Some constraints use slot values that they do not really "depend on." For example, a constraint checking the appropriateness of target and aircraft could optionally use the unit slot if the unit had only a single type of aircraft. This constraint would formally depend on all three slots, but would only use two, depending on whether the aircraft slot had a value or not.

What would happen if constraints reported which slot they really depended on? That is, suppose each constraint that failed gave a list of the slots whose values were implicated. Then any constraint that depended on slots implicated by a higher priority constraint could safely be elided from the list of constraint failures told to the user. Although the experiment has not been performed, we suspect the results of this scheme and those of constraints correctly annotated by timeliness specifications would be identical.

We therefore believe that the constraint timeliness mechanisms pre-compute a result that could be obtained quite simply by an after-the-fact analysis of constraint failures.

COMPLEX CONSTRAINTS

The need for coordination between two or more missions is most obvious when in-flight refueling is called for. Refueling missions, in fact, can service more than one OCA mission. The refueling aircraft flies in an oval path called an "orbit", where it waits for a rendezvous with one or more missions. The refueling aircraft may move to a second orbit at a different location after a scheduled time.

It is convenient to allocate a whole FRL frame to each orbit, so that parameters like the orbit start time and location can be kept together. Mission information like time of departure and home airbase of the refueling aircraft are stored in the refueling mission frame. For similar reasons, each refueling service within an orbit also has its own frame.

When the planner selects "REFUELSVC" as the item to fill in the mission menu, the \$PROMPT procedure in the corresponding slot of the template re-enters the mission-planning level with a new menu, one to build a service frame representing the plan for the OCA and refueling mission to meet. The orbit containing the service, and the refueling mission itself, also have to be planned. Each has its own template and its own menu. The planner can use control commands to move back and forth between the frames being filled until they are all complete.

The relationships between an OCA mission frame, a refueling (RFL) mission frame, an orbit (ORB) frame, and a service (SVC) frame are depicted in Figure 14. The frames are tied together with pointers, i.e., slots whose values are names of other frames. Each pointer has an inverse, just as AKO is an inverse to INSTANCES. These pointers and inverses are needed to locate the arguments of multiple-frame constraints.

Multiple-frame constraints make use of remote arguments. An example constraint is RANGE2, which determines whether the aircraft in an OCA mission can reach the scheduled refueling orbit on time. This is how it appears in the OCA template:

```
(RANGE2 (AIRBASE AIRCRAFT TARGET TOT REFUELSVC
        (ORBLOC REFUELSVC ORBMSN ORBLOC)
        (RTIME REFUELSVC RTIME))
  (*AND* AIRBASE ... ORBLOC RTIME))
```

By following the path in the first remote argument, we see that the orbit location ORBLOC is an item in the orbit frame.

A multiple-frame constraint has to be listed in the template for every frame having slots that are arguments in the constraint, since a change in any of those slots must trigger a constraint check. The constraint also must be listed in the template for every frame having a slot that appears in a remote path in the

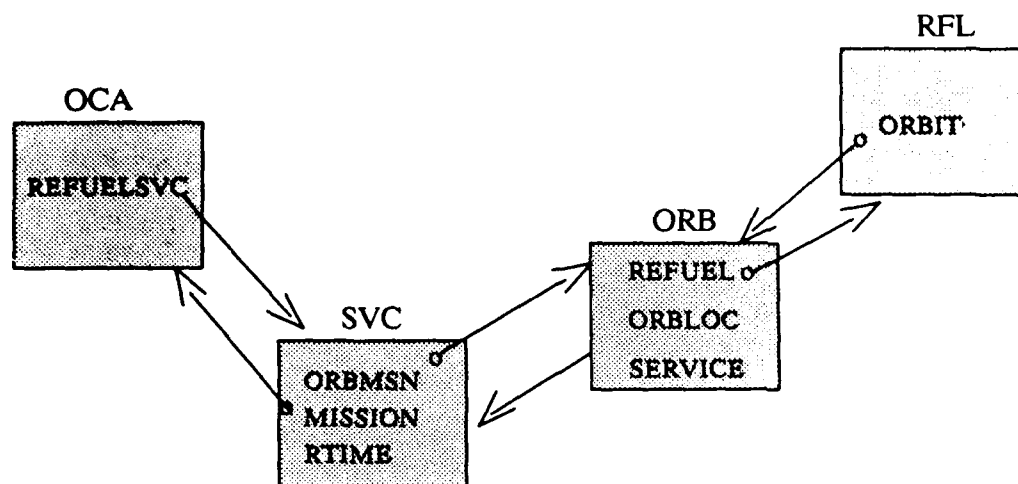


Figure 14. Relations Between A Mission, A Refueling, An Orbit, And A Service.

constraint reference, since a change in a pointer along the path means that a different argument value (in a different frame) must be tested.

Normally an argument access path is interpreted relative to the frame in whose template the constraint appears. This means that when the same constraint reference is put in another of the related frames, the access paths are no longer valid. To obviate rewriting the access paths relative to the other frame, a means has been provided for shifting the frame of reference (pun intended) instead. By putting the flag (*MASTER* access-path) in the constraint reference, the frame reached by the given access path will serve as the base frame for getting both local items and remotely accessed ones. When RANGE2 appears in the ORB template, for example, it looks the same as above except for the *MASTER* flag:

```

(RANGE2          ;;This is the name of the constraint
 (AIRBASE ... (RTIME REFUELSVC RTIME)) ;;This is the list of slots
 (*AND* AIRBASE ... RTIME) ;;Timeliness condition
 ((*MASTER* SERVICE MISSION))) ;;and finally the flags.

```

The SERVICE-MISSION path takes us from the ORB template to the OCA template, where the given arguments AIRBASE, etc., are obtainable.

THE NATURAL LANGUAGE SUBSYSTEM

Natural language interaction has been an important theme in KNOBS from the beginning. Indeed, the developers estimate that over 60% of the development went into direct support of natural language; interface issues dominated -- and at times monopolized -- the KNOBS research and development effort.

At any time during the development of the system, from MICROKNOBS to the present, one could ask questions in English about the contents of the data base. Mission planning was supported at a low level in MICROKNOBS by the command, "CHOOSE MUNITIONS FOR ...", and now a complete mission can be planned through an English dialogue. The rule editor also makes use of the natural language subsystem, since new hypotheses are entered in English.

Three distinct natural language understanding mechanisms have been used in KNOBS. First was the pattern matcher in MICROKNOBS. This was replaced by an ATN (Augmented Transition Network) parser in early versions of KNOBS. Finally, the current system has a conceptual dependency parser. The first two will be sketched briefly; the final one will be described in greater detail.

Translation of rules to English and explanation of inferences do not require the use of the parsing apparatus; they are handled by a different mechanism more like the one originally provided in MICROKNOBS. The present form of this mechanism is described at the end of this section. Most of the output and explanations produced by KNOBS are "canned" (rule translation is the only notable exception). The English generation capabilities in KNOBS are only a very small part of the system.

WHY NATURAL LANGUAGE?

To some, it may seem self-evident that an expert planning system needs a natural language interface. How else could a user possibly exercise all of the capabilities of the system? As it turns out, the answer is "quite easily."

What can the user ask of KNOBS? The user can ask for any property of any entity in the KNOBS data base (stored as frames, as described in earlier chapters). But there are relatively few properties of any frame in KNOBS (maximum of around a dozen). The user can request something be shown using the rule system. But there are very few things the rule system can actually show (again, maximum of around a dozen). The user can fill a slot of one entity with another: there are hundreds of entities in the frame data base. The user can restrict the contents of a slot by, for example, giving a numeric range or specifying a subset of possible candidates as the only acceptable ones (more general restrictions were not implemented). Finally, the user can exercise the capabilities of the KNOBS inference mechanisms: get documentation of slots that need to be filled, request a list of candidates for a slot, ask for list to be ordered, attempt to fill a slot, and ask for explanations of constraint failures and ordering criteria.

It would not be difficult to invent a formal language that could express any question or command to KNOBS: using LISP syntax, the language would probably consist of 30 or so commands, plus all the names of things and the names of properties of things in the data base. One suspects that learning this hypothetical KNOBS command language would not be much of a strain: a few hours of training would be more than sufficient. The training time for this hypothetical language could be reduced to zero by embedding the command language into a set of menus and on-line help.

"Menus" Were Not Available

Remember that KNOBS was operating on a remote time-shared computer using the ARPANET. Although a high speed network, the actual terminal to program character rate was slow: achieving a line-a-second throughput was extremely rare. This slow communication would have been enough to prohibit use of menus. But even worse, KNOBS was a large system that would usually be swapped out of the time-shared computer while waiting for user input. The swapping overhead probably dominated the actual CPU time.

The factors above, beyond making a menu-controlled system unthinkable, also contributed to the desire to minimize KNOBS's interaction with the user to a minimum.

The Hypothetical ENUMERATE Command

Let's continue our supposition that a small command language had been implemented for KNOBS. It might include a command

(ENUMERATE source [predicate] [accessor]).

"Source" is either the name of a frame, in which case all instances of that frame (and any other frame that is a subclass) is taken as the set being enumerated, or some list of possibilities. "Predicate" is a form to be evaluated with the symbol for the "source" bound to a possibility (by default, the predicate is the constant "true" function). If the predicate returned non-NIL, then the "accessor" form would be evaluated to get the thing returned as the value of the enumeration, which could either be a set or a single thing. Finally, the result of the enumeration is kept free of duplications.

In order to get the values of slots, the hypothetical command language might include a form

(SLOT slot-name frame)

where both "slot-name" and "frame" can be sets instead of singletons. The form would return a duplicate-free list of entities in the specified slot (or slots) of the supplied frame (or frames).

For example, with AIRBASE bound to some airbase, we would say something like

(ENUMERATE (ENUMERATE 'UNIT (AT UNIT AIRBASE))

T

(ENUMERATE (SLOT 'AIRCRAFT UNIT)))

to get the aircraft at some airbase.

The Need for Question Patterns

We might soon decide that we would like to ask for a list of the aircraft at an airbase directly, rather than having to use the rather long expression above. We might want to map a "question" of the form

(ENUMERATE (SLOT 'AIRCRAFT AIRBASE))

into the "official" form. Notice that the user is freed somewhat from having to know the structure of the frames data base.

Such a translation from one command into a longer, more involved, but semantically equivalent form is the simplest service provided by a natural language interface. Along with translation of one command into another, these "question patterns" would allow a certain form of synonym to be used. We could say that whenever we saw a slot names 'A/C, it should be replaced by the name 'AIRCRAFT.

The Need for Ellipsis and Anaphora

Two airbases in the KNOBS data base are "Hahn" and "Ramstein." The user could ask (again, in this hypothetical command language)

(ENUMERATE (SLOT 'AIRCRAFT 'HAHN))

to get a list of the aircraft available at Hahn. Instead of typing all those parentheses again, the user might want to be able to get away with just typing

'RAMSTEIN'

and getting the command interpreter to realize that what the user meant was that the previous query be re-computed with the only thing in the query that could be replaced with an airbase replaced with Ramstein.

The paragraph above can be summarized as showing that the user wants to use "ellipsis": a portion of the query is simply not given the second time.

Anaphora, in which a pronoun or other phrase is used to stand for a previously given complete description, is closely related to ellipsis. I would want the command processor to deal with anaphora if I wanted it to supply an interpretation for the hypothetical command

(ENUMERATE (SLOT 'SCL *them*))

by noting the only thing that I have recently mentioned that have SCLs are those aircraft at Ramstein, and substituting that list into the query above for the "pronoun" *THEM*.

The Need for a "Natural Language" Interface has Nothing to do With "Natural Language"

We began this discussion by trying to answer the question, "Why does KNOBS need a natural language interface?" We argued that even if the high-resolution bit mapped screen and pointing interface had been available, there would still have been a need for additional inference mechanisms. These inference mechanisms then turned out to be precisely those provided by a natural language subsystem.

The discussion above has made the sneaky argument that a "natural language" interface should supply certain capabilities, and that those capabilities are not closely related to specific phenomena in English or any other natural language. Of course, there are specific linguistic concerns in a natural language interface (e.g., word morphology). But a natural language interface is desirable because of the inference mechanisms it brings to bear, rather than because its "raw command syntax" happens to coincide with the user's native tongue.

The capabilities explored in the KNOBS project are as follows:

1. Pattern-driven re-phrasing of questions and commands
2. Anaphora and Ellipsis
3. Synonym substitution
4. Context-sensitive word sense disambiguation
5. Fill-in inference (e.g., if asking if an aircraft can fly to a target assume that the aircraft begins its flight at its "home" airbase)

There are a number of other phenomenon that KNOBS did not try to address. These include generation of natural language responses (except for explaining rules, as described below), integrating natural language with other forms of input (e.g., menu choices), and other forms of interaction (e.g., charts, graphs, maps, tables). Nor did the natural language subsection of KNOBS attempt to deal with discourse phenomenon (e.g., if I ask the same question twice in a row, I certainly do not want the same answer the second time; KNOBS did not track discourse, so it would not even "know" that I had asked the same question again, much less have any inference mechanism that would be able to figure out what to do differently).

THE MICROKNOBS PATTERN MATCHER

The simplest way to build what appears to be a natural language capability is to use a simple pattern matcher. A famous example of this type of natural language interface is the DOCTOR program which, when all else fails, asks the user (more properly, the "victim") to "TELL ME ABOUT YOUR MOTHER."

The entire pattern matcher can be contained in a few pages of LISP code; writing one from scratch is a good 1-week homework assignment for

undergraduates in an introductory AI course. Despite their simplicity, with a fairly rich set of patterns, the performance of such interfaces can be surprisingly impressive.

MICROKNOBS used pattern/action production rules to respond to English input. Prior to pattern matching, the input was standardized by removing punctuation, substituting preferred synonyms for significant words, and unifying certain phrases into hyphenated atoms. The phrases TARGET 1 and CHOOSE MUNITIONS, for example, were turned into the atoms TARGET-1 and CHOOSE-MUNITIONS.

A production rule had a pattern, some tests, and some LISP expressions to evaluate if the pattern and tests are satisfied. It had the same format as a back-chaining rule with one hypothesis. This is the rule that handled a CHOOSE MUNITIONS command:

```
((((?CMD ! ?TARGET)
  (?CMD (OR (EQ ?CMD 'CHOOSE)
    (EQ ?CMD 'CHOOSE-MUNITIONS)))
  (?TARGET (TARGETNUMCHECK ?TARGET)))
  ((SETQ LASTCOMMAND 'CHOOSE-MUNITIONS)
  (SETQ TARGET ?TARGET)
  (SANSPARENPRIN1
  (EVAL (CONS 'CHOOSE-MUNITIONS (LIST TARGET))))
  (TERPRI)
  (SETQ CHEXPL EXPLANATION))))
```

When the pattern (?CMD ! ?TARGET) in this rule is matched with the input "CHOOSE MUNITIONS FOR TARGET 1", the variable ?CMD receives the value CHOOSE-MUNITIONS. The exclamation point is a don't-care symbol for skipping over words. The pattern matcher would first try FOR for ?TARGET, but TARGETNUMCHECK would fail; hence FOR is skipped and TARGET-1 succeeds for ?TARGET.

IF this rule succeeds, its third action calls the function CHOOSE-MUNITIONS, which in turn calls INFER with a BEST-MUNITIONS clause.

The pattern matcher was furnished with enough patterns and local memory to recognize almost any request the system was capable of answering. The problem with a pattern matcher is that it misinterprets some input, because the words it ignores can totally alter the meaning of the input. For example, after a question about TARGET 2 is answered properly, the question "WHICH TARGET IS AN AIRBASE" receives the response "TARGET 2 IS A TANK BATTALION". The question fits the pattern (! ?TARGET IS-A ?TYPE).

Discussion of the Pattern Matcher

The pattern-matching approach outlined above does not address any of the capabilities that we want in a natural language interface, except for accepting input that looks like English.

The pattern-matcher was intended to hint at the desirability of natural language interaction, and to serve as a place-holder for a more sophisticated natural language capability.

THE ATN PARSER

In 1980, the pattern matcher was supplanted by an ATN parser. An ATN parser makes a real effort to "understand" a sentence in the way people are taught. It determines the part of speech for each input word - noun, adjective, etc. - and identifies noun phrases, the subject and object of the sentence, and so on. The network is a way of encoding an English grammar. For an explanation of natural language parsing using an ATN, see [WOOD70].

The augmented transition network itself was only one part of the input processing. There were four steps, depicted below.

A morphological analyzer attempts to identify individual words in the input, despite the presence of suffixes, spelling errors, or irregular inflections. It uses a dictionary to associate words with such information as parts of speech, tense, pluralization, and occurrence in an idiom. In KNOBS, the dictionary entry for a word was part of its property list.² The frame data base acted as an extension of the dictionary - the name of any frame was taken to be a noun. If the frame had an AIO entry, it was a proper noun, otherwise it was a common noun.

The syntactic analysis performed by the ATN resulted in the assignment of values to certain "registers" to record the essential information carried by the sentence for use by the semantic components. As an example, the sentence "WHAT KIND OF CONTROL RADAR DOES BE50362 HAVE" results in the register assignments below.

Register	Contents

SUBJ	(BE50362)
OBJ	((((AIO AKO) ?
CONTROL-RADAR))	
V	HAVE
QTYPE	WHAT-TYPE

The subject, object, verb, and question type have been identified. The object is really a noun phrase: "a kind of control-radar". It is represented in roughly the form of a rule clause, with a question mark for the unknown frame.

The semantic components, for noun phrase resolution and question answering, include various strategies to answer queries or perform requested actions, such as printing rules or modifying the data base.

Because the ATN parser is a machine for recognizing a specified grammar for English, it could not handle ungrammatical utterances, or uncommon constructions that had simply not been incorporated into the grammar. It had particular difficulty with sentence fragments, which are common in conversation, such as "AND ITS SPEED?" after a previous query about an

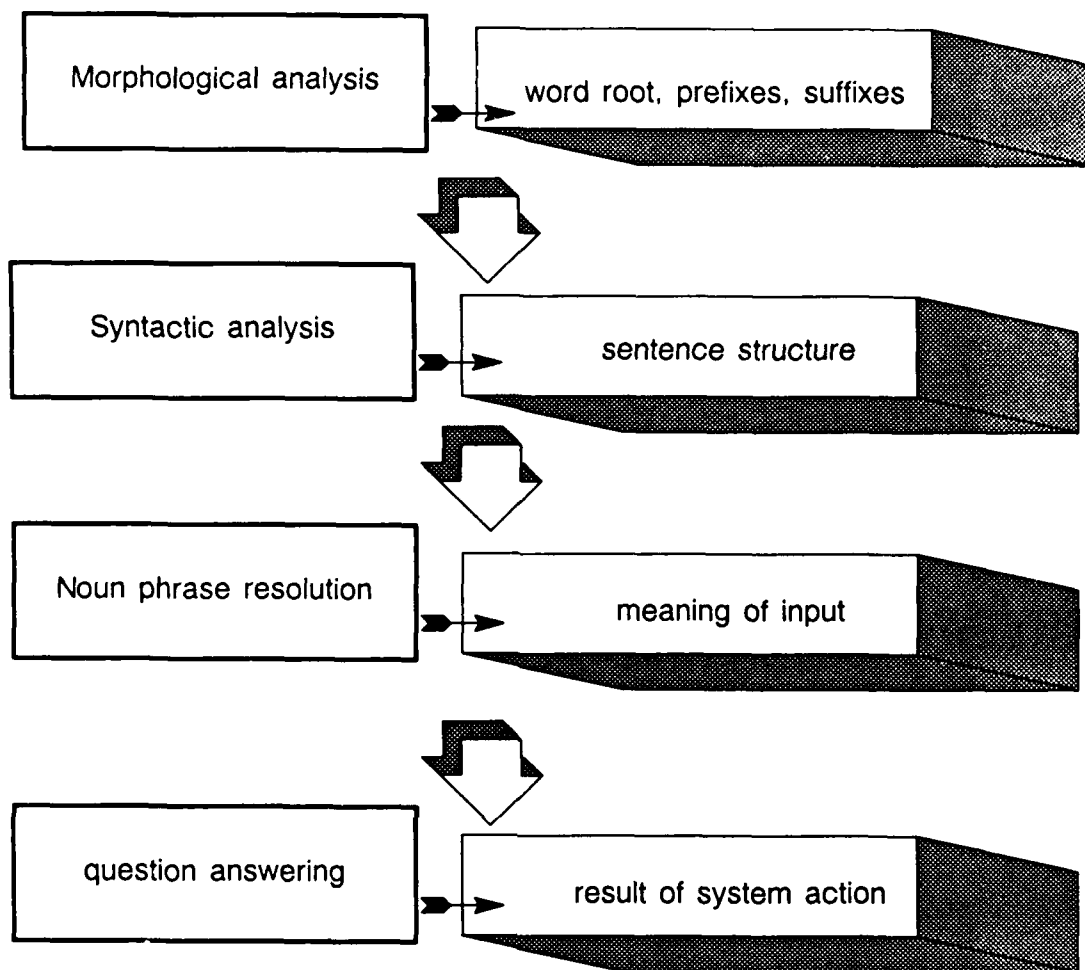


Figure 15: From Surface Structure To Understanding

aircraft. The flexibility of a pattern-action production rule system was also missing.

Discussion of ATN-Based Natural Language

An ATN is based on a transition network (more precisely, a finite-state machine) that "accepts" grammatically correct inputs. Expressing a grammar as an ATN consists of building a machine that recognizes acceptable inputs, as compared to a "transformational grammar" that describes acceptable inputs by giving a way to generate all and only those inputs. The difference between an ATN and a traditional finite-state machine is that on transitions from one state to another, the ATN can "jump to" or recursively invoke another finite-state machine.

The claim that an ATN cannot accept "ungrammatical input" is somewhat misleading. In fact, for any list of acceptable inputs (which may or may not correspond to traditional English) there is a grammar. The difficulty is that the class of inputs that were supposed to be acceptable for KNOBS could not be described by a grammar (again, the grammar for English is just one of an infinite number of possible grammars) easily expressed using ATNs. The primary difficulty seemed to be that in order to interpret nongrammatical inputs (e.g., sentence fragments), the dialogue context and possible referent information needed to be used. But the ATN formalism did not provide a place for that information. Thus ATNs did not appear suitable. It is meaningless to try to talk about a grammar without having some formal means of expressing that grammar; ATNs simply do not seem to be a good way to express the grammar for the inputs KNOBS needed to understand.

To summarize, the reason the KNOBS researchers gave up trying to express the desired KNOBS grammar using ATNs has to do with the enormous context-sensitivity required to tell if a random word or phrase was to be acceptable in the given discourse context. Whether the acceptability of such an utterance could or could not be expressed by ATNs is moot.

THE CONCEPTUAL DEPENDENCY PARSER

In 1981, the ATN system was replaced by a conceptual dependency parser. The parser, called APE-II, operates differently from the prior ATN parser in KNOBS, or those in such familiar natural language query systems as LADDER [HEND78] and LUNAR [WOOD72]; it is driven by word meanings instead of sentence syntax, and aims at a deeper meaning representation. The conceptual dependency idea was originated by Schank [SCHANK72].

The semantic component was replaced by a new set of pattern/action production rules. Patterns were not matched with the input, but rather with the "concept" produced by the parser as the meaning representation of the input. The production rules are operated by the responder component of the system, which will be described in the next subsection.

Concepts

A conceptual dependency parser turns sentences into concepts. A concept is a frame-like structure that consists of a type and a collection of labeled slots.

For example, the sentence "THERE ARE F-4C'S AT HAHN" is parsed into the concept:

(*EXISTS* OBJECT
 (F-4C NUMBER (*PLURAL*))
 LOC (AT PLACE (HAHN)))

of type *EXISTS*, stating the existence of an object. It has two slots, OBJECT and LOC, which are referred to as cases (or roles, or gaps). The values given under cases, called case fillers, are themselves concepts, given in full or in part.

In the example the two case fillers are concepts of types F-4C and AT, each with one case included.

The most general concept types for sentences are *ACT* and *STATE*. Below these are more specific concepts, forming hierarchies illustrated in part in Figure 16. The hierarchical relationships are supported by FRL using AKO links. Some common primitive acts are *PTRANS*, transfer of physical location (go, walk, etc.); *ATRANS*, transfer of ownership (give, buy, etc.); and *INGEST*, taking of an object into the body (eat, inhale, etc.).

Physical objects also have concepts, called picture producers. Their types also form a hierarchy below the most general type *PP*, ranging from general concepts such as PERSON and VEHICLE down to specific types like F-4C.

The concept created for a question is just like that for a statement except that there is a question concept *?* in place of a case filler which is being asked about. For example, the question, HOW MANY FIGHTERS ARE AT HAHN? is parsed into the conceptual dependency:

```
(*EXISTS* OBJECT
      (NUMBER ARGUMENT
        (FIGHTER NUMBER (*PLURAL*))
        VALUE (*?*))
      LOC (AT PLACE (HAHN)))
```

The Parsing Procedure

Words in the sentence are processed from left to right. As a word is processed, a lexical entry for it is added to a list called the concept list. A lexical entry is an atom whose property list contains the word it came from and a copy of the dictionary definition of that word.

The dictionary definition of a word contains one or more senses for it. For each sense of the word there is a concept template – a concept with some case fillers equal to (NIL), indicating that they are to be filled in with concepts obtained elsewhere in the sentence.

Associated with each sense is a set of expectations, which are conditional actions causing a variety of possible effects; the simplest is to cause (NIL) cases in other lexical entries to be filled. There are different kinds of expectations: (ordinary) expectations, lexical expectations, and structural expectations. The last kind are associated with a word rather than with any particular sense of it. Expectations operate not only on the concept list but also on "active memory", an association list containing useful data such as the "current concept," "focus," and concepts saved from the last sentence.

The next step in parsing, after adding a new lexical entry for a word to the concept list, is to attempt to satisfy its expectations, and the as-yet unfulfilled expectations of other lexical entries.

The effect of fulfilling expectations is to remove lexical entries when one or more of their senses have been used to fill (NIL) cases in other lexical entries.

After the expectations of the last word in the sentence are fulfilled, there should be only one lexical entry left, with one sense, arising from the main verb in the sentence. If the sentence is ambiguous or incomplete, this may not happen,

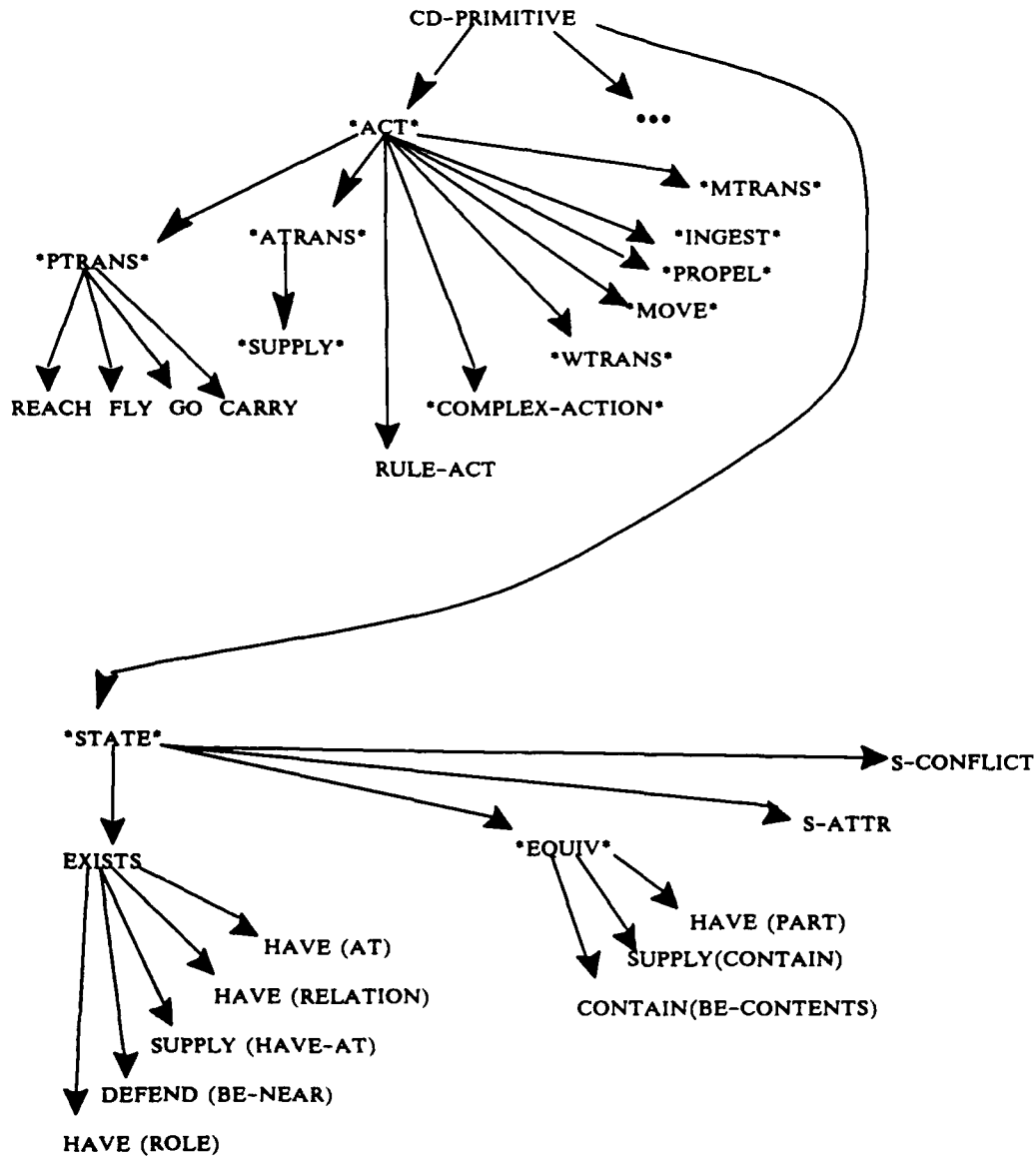


Figure 16. Part of the Hierarchy of Primitive Concepts

and the program does the best it can to interpret what is there. It prints "CAN YOU PLEASE REPHRASE THAT?" if all else fails.

Dictionary Definitions

Words which are represented as picture producers typically have very simple definitions because they usually have no expectations. The word "woman," for example, has a definition which builds a picture producer whose type is PERSON and whose gender is FEMinine:

```
(DEF (SENSE (PERSON GENDER (*FEM*))))).
```

Slightly more complex are words which fill a case in a picture producer. For example, the definition of the word "the" creates the concept DEFinite. It expects a concept which could be a picture producer or a relation to modify. When this concept is found, its REF slot will be filled with DEFinite.

```
THE:
  (DEF (SENSE (*DEF*)
    EXPECTATIONS
    ((IF (MODIFIES (OR *PP* *EVENT*
      *RELATION* *SCRIPT*))
      THEN ((SLOTS (* REF)))))))
```

Anaphoric References

An anaphoric reference is a word or phrase that refers to an object or concept mentioned in a previous sentence or implied by context. Pronouns are anaphoric, also phrases like "this place" or "the target." References that may be resolved by local context are handled, in this system, by saving the concept produced for the last sentence. References that must be resolved in a global context are handled by the same pattern recognition scheme that is used to generate responses.

Arbitrary Decisions in Conceptual Dependency

Given our current understanding of how information can be represented within a computer, we must admit that any scheme for representing knowledge involves some arbitrary decisions. However, our experience with KNOBS leads us to conclude that the so-called "theory" of conceptual dependency involves more arbitrary decisions than others.

One source of the ad hoc quality of CD theory is its dependence on primitives. Schank began with eleven primitive concepts which he claimed represented language universal notions; over the years, however, Schank has found it convenient to expand the list of primitives, usually to process some new body of texts dealing with a particular domain such as politics or train wrecks. The fact that Schank has responded to new domains in this way does little to inspire confidence in his choice of primitives; furthermore, although he claims the universality of these concepts he presents no convincing experimental, anthropological, or psychological evidence to support his claim. Given the lack of such evidence, plus the fact that practitioners of CD theory have dealt with

only a tiny fraction of texts, most of them English, one is forced to conclude that the choice of these primitives was an arbitrary decision on the part of Schank and his followers.

CD theory forces its practitioners to make other arbitrary decisions as well. One of Schank's primary considerations in developing CDs as a knowledge representation scheme was to capture the relations of consequence, instrumentality, and causation that link actions together. For example, he describes the CD network for the sentence "John gave Mary the book" as a transfer of possession involving the use of John's hand as an instrument in the movement of the book from John to Mary. Schank noted in his description of this representation that according to this representation it might never be possible to finish diagramming a concept involving instrumentation. He states "Conceptually we might have something like: John ATRANSed the book to Mary by moving the book towards Mary, by moving his hand which contained the book towards Mary, by grasping the book, by moving his hand, by moving his muscles... ." The CD implementer must make a decision as to the level of detail he wishes to capture; such a decision cannot be other than arbitrary. Schank does not point out here that the question of "how far to go" is not limited to the instrumentality relation but applies to other relations as well. Presupposition, for example, could be handled by adding the presupposed states to a CD representing a given action, but it would be up to the implementer to decide which presuppositions, if any, to represent.

Yet a third problem for this theory is its admitted lack of a specified semantics. No one has produced such a specification, and students of CD are forced to learn by example and by direct teaching rather than from a canonical source for CD semantics. This is a particular problem for anyone who attempts to deal with such issues as quantification within a CD framework since understanding the role of quantifiers forces a researcher to specify exactly what he means in formal terms. Mapping such a specification to a system in which the semantics must be inferred is clumsy, difficult, and risky in the sense that one never knows if one has gotten it right.

THE RESPONDER

Responses are generated by matching sentence concepts to patterns, and executing the action associated with that pattern. Patterns are concepts with variables in them; the pattern-matching process binds the variables to subconcepts that appear in the concept being matched. The binding of a variable is subject to a predicate associated with that variable. Also, if the variable has already been bound, the match is successful only if the previous binding is compatible with the attempted binding.

The type of the pattern concept does not have to be matched exactly; patterns for more general concepts, i.e., those above the concept to be matched in the hierarchy, are applicable.

Patterns are associated with actions in production rules for questions, commands, and statements. There are also production rules that aid in the

parsing process, to resolve references to global context and to perform inferences. Question-answering, inference productions, and rule editing will be discussed below as representative examples of well-developed capabilities. The same approach is also used to provide the planner with an alternative to the menu approach for interactive mission planning; that will also be illustrated.

Question-Answering

There are about two hundred question-answering pattern-action production rules in KNOBS. The question-answering patterns are grouped according to the domain for which they are applicable and the focus of the question. In general, there will be a small number of patterns which seem applicable to a question concept. These patterns are prioritized, so that the action of the first pattern which matches a question concept is executed. An example pattern-action production, with comments in place of some of the actual LISP code, is:

```
(PAT (*PTRANS* TO (*PROX* PLACE &TARGET)
FROM (*PROX* PLACE &AIRBASE)
OBJECT &AIRCRAFT
MODE (*POTENTIAL*)
Q-FOCUS (MODE)
ACTION [get list of aircraft that can reach &TARGET
        from &AIRBASE]
SCRIPT OCAT
RESPONSE [print yes if &AIRCRAFT is on the list, else no]
NAME CAN-A-PLANE-FROM-AIRBASE-REACH-TARGET
FIELDNAME OCAT).
```

This pattern is designed for answering questions such as "CAN AN F-15 REACH MERSEBURG-F FROM BITBURG?" The question focus is a case or case path (to reach a nested concept) which would lead to the "?? entry in the concept being matched. In this pattern, the focus is MODE, which indicates it is a yes/no type of question. The pattern contains three variables, indicated by "&"-prefixed names: &AIRCRAFT, &TARGET, and &AIRBASE.

The script identifier is used to limit the search for patterns. A script may be thought of as a set of patterns that are appropriate for a given context, such as a particular type of mission. At any time there are only one or a few active scripts, and only those production rules identified with an active script need be considered.

Inferences

In the process of trying to answer a question, it may be discovered that no production is applicable. Sometimes a potentially applicable pattern exists, but no bindings are given for one or more of its variables. In this case, a "conceptual completion" inference is sought to find them. At other times, no applicable pattern is found. In such cases, a "causal" inference is attempted to find a different, answerable question that is implied by the original question, and therefore may have been the intent of the asker.

Conceptual completion inferences are accomplished with the help of a specialized type of pattern-action production rule. Here is an example:

```
(PAT (*PTRANS* OBJECT &OBJECT)
  ACTION (FIND-LOCATION &OBJECT)
  INFERENCE (FROM))
```

This rule is designed to infer the missing source, or FROM case, of a physical transfer. It binds the variable &OBJECT to the filler of the OBJECT case. The action finds a location associated with the object, either by looking in the LOCATION case of the &OBJECT concept, or by checking the data base for the known location of the referent of &OBJECT.

This rule would be unnecessary to process the question "WHICH AIRCRAFT AT RAMSTEIN COULD REACH THE TARGET FROM HAHN?" because the "from" location of the transfer is explicitly stated. It would be needed, however, for "WHICH AIRCRAFT AT RAMSTEIN CAN REACH THE TARGET?" in order to infer that the "from" location is Ramstein.

Causal inferences connect the states and actions of a stereotypical situation. They are represented by causal links [CULLINGFORD78] in a set of script-associated production rules called "scenes". Instead of an ACTION, these rules have RESULT, ENABLE, or RESULT-ENABLE attributes which identify the causally related patterns. Actions RESULT in states, and states ENABLE actions. When an action enables a state, it may have a RESULT-ENABLE link to actions enabled in that state.

For example, the question "WHICH AIRCRAFT AT HAHN CAN STRIKE BE70701?" yields a conceptualization that does not match any question-answering pattern, but does match the pattern in the following scene:

```
(NAME AC-HIT-TARGET
  PAT (*PROPEL* ACTOR &OCA:AIRCRAFT
      TO (*LOCSPEC* PLACE
          &OCA:TARGET)
      OBJECT &OCA:SCL)
  SCRIPT OCA
  RESULT TARGET-IS-DESTROYED
  AFTER AC-FLY-BACK)
```

The &OCA:TARGET variable is bound to BE70701, the &OCA:AIRCRAFT variable gets (AIRCRAFT IS-A (*?) LOC (AT PLACE (HAHN))), and the &OCA:SCL variable is unbound.

The scene-chaining interpreter looks forward and backward in the script's causal relationships to find an implied pattern that can be both matched and answered. In this case, it finds the AC-FLY-TO-TARGET action, which has a RESULT-ENABLE link to the AC-HIT-TARGET action.

```

(NAME AC-FLY-TO-TARGET
PAT (*PTRANS* OBJECT &OCA:AIRCRAFT
    TO (*PROX* PLACE &OCA:TARGET)
    FROM (*PROX* PLACE &OCA:AIRBASE))
SCRIPT OCA
AFTER AC-HIT-TARGET
RESULT-ENABLE AC-HIT-TARGET
RESULT AC-OVER-TARGET)

```

When the variables are bound with the same values, this pattern represents the question: "WHICH AIRCRAFT AT HAHN CAN REACH BE70701?" and there is a question-answering pattern for that one.

Applications of CD-based Natural Language

Rule Editing

Rules are flexible as well as explainable; that is their principal advantage as a means of knowledge representation. Rules can be modified; furthermore, new rules can be added, and old ones deleted. The KNOBS rule editor permits the hypothesis of a rule to be entered in English. The conclusions are limited to a few stock types, so they are entered with a multiple-choice technique.

There are pattern-action productions whose purpose is to convert the English form of rule hypotheses into their internal form, to support rule editing. An example of one of these is the following:

```

(PAT (*EXISTS* OBJECT &WPN
    OBJECT (WEAPON)
    LOC (*PROX* PLACE &TARGET))
ACTION (ADD-RCLAUSE
    (THREAT-TYPE (RGET-CON-REF &WPN))
    (ENTER-RULE-VAR &WPN
    (MAKE-RULE-VAR (RGET-CON-REF
        &WPN)))
    (RGET-CON-REF &TARGET))
NAME IF-X-DEFENDS-TARGET)

```

This version of the "DEFENDS" production is obsolete - the current version is much more complete - but the way it works is more easily comprehended from this simpler version.

As the NAME case suggests, this pattern handles hypotheses like "THE TARGET IS DEFENDED BY THE SAM", and produces a clause like (SAM-THREAT ?SAM ?TARG). Implementing this facility required writing some special-purpose functions like ADD-RCLAUSE for constructing the desired result. Existing functions like RGET-CON-REF were handy for looking in a concept to find a frame that it refers to, or a variable representing that frame.

Truth Maintenance

Constraints have to be checked or rechecked when a mission item changes, or when there is a change in the data base; the remaining possibility is that the constraint itself has changed. This can happen, for rule-based constraints, when the rule is edited.

KNOBS maintains a network of cross-references which represent the dependencies which exist between rules. Rule names and keywords have property list entries indicating occurrences of keywords in rules; two rules are linked if one has a hypothesis whose keyword is established as a conclusion of some other rule. When a rule is added, deleted, or modified, the network of property list links is updated as necessary, and it is traced upward to determine the rules and consequently the constraints that may have been affected, and these are re-evaluated in the context of the current missions. The network has the side benefit that it detects any co-dependency or circularity in the system of rules.

Truth maintenance checking shows up in messages displayed after the user has finished editing a rule. The following message might appear:

Re-evaluating the constraints affected by
TARGET-AIRCRAFT-2

Rechecking frame OCA1001.

NEW CONFLICT.

The following missions have newly-created
conflicts because of the rule changes:
OCA1001.

Instead of NEW CONFLICT, the checking may find that a conflict has been eliminated by the change, and will list the missions with "reduced conflicts"; or it may find that no missions are affected.

Dialogue-Based Planning

The following conversation with KNOBS illustrates the entering of plan components through the English subsystem:

USER: SEND 4 AIRCRAFT FROM THE
109TFW TO STRIKE BE70501 AT 0900.

KNOBS: What AIRCRAFT do you want to
use?

USER: WHAT AIRCRAFT ARE IN THE
109TFW?

KNOBS: The 109TFW has F-4Cs. Would
you like to use F-4Cs for the AIRCRAFT?

USER: NO, F-4G'S.

KNOBS: The 109TFW does not contain
F-4Gs.

USER: FLY THE F-4G'S OUT OF THE
126TFW AT RAMSTEIN.

There are productions in the OCAT script (mentioned above under Question-Answering) whose actions fill the cases of an OCA mission concept. The cases of that concept correspond to the items in an OCA mission frame. The individual OCA mission will be created if necessary, and its item slots will be filled in with values corresponding to the case fillers.

The first user sentence, for example, fills the TARGET case with the concept (BE70501), and the TARGET item slot in a new OCA mission with BE70501. When slots are filled, the constraints are triggered as usual, and failure reports are explained to the user; this is what happened when F-4G's were suggested.

Productions in the OCAT script that respond to planning commands have actions that prompt the user to fill in items that have not yet been specified, such as the aircraft type in the first response. At any time, the user can exit from the English mode and return to the menu to complete the mission.

TRANSLATION OF RULES TO ENGLISH

Translating the internal form of a rule into readable English has two interesting subproblems: one is the translation of LISP expressions into English, and the other is the condensation of statement sequences. Condensation means to combine two statements into a single one by eliminating repeated references. For example, "The target is X" and "X is a runway" can be condensed into "The target is a runway."

Translating LISP Expressions and Clauses

English production from LISP is not hard; the general idea is to supply a fill-the-blanks kind of template for each function. For example, an expression like (LESSP A 5) becomes "A IS LESS THAN 5." The expansion process must be recursive to handle nested expressions occurring as side conditions, like (LESSP (MAX A B) 5): "THE MAXIMUM OF A AND B IS LESS THAN 5."

The same idea is used to expand clauses; the template is associated with the keyword. The keyword CHECK-TARGET-AIRCRAFT, for example, has a template that begins "THERE IS A ___ CONFLICT BETWEEN...", where the blank is filled in with "MODERATE" or "SERIOUS" depending on whether the second constant in the clause is "*" or "**"

Each keyword or function that is used in rules is provided with a function that yields its English translation. The translation function for a keyword has three arguments: KEY, OBJ, and VAL, which are associated with the three elements

in a clause: the keyword, an object, and a value. The translation function for a LISP function has arguments corresponding to a typical function call.

If the elements of the clause or expression are constants, they are supplied as the arguments to the translation function. If an element of the clause is an expression (like (MAX A B) in the example above), that expression is recursively translated into a list of words. The translation function of the outer clause is then called with the translation of the inner expression for the appropriate argument.

Condensation

There is a rule having the following hypothesis in its printed form:

THE TARGET OF THE ?MSN RADIATES

This is really the condensation of two hypotheses in the internal form:

((TARGET ?MSN ?X))
((IS-A ?X ELECTRONICS))

The RULETRAN function recognizes an opportunity to combine these two clauses into a single sentence because of the shared variable ?X.

Since there is a variable in the clause that does not have a binding, there is no translation for the clause; instead, a translation for the variable is created. It translates ?X because ?MSN is recognized as occurring in the conclusion of the rule, and cannot be eliminated in translation. The clause (TARGET ?MSN ?X) gives ?X the translation (THE TARGET OF THE ?MSN). This translation comes from another function on the property list of the keyword TARGET. The translation for ?X is not printed immediately; it is just bound to ?X on an association list, where it is saved for later use.

Now, when another clause comes along with ?X in it, the translation function for IS-A substitutes the binding for ?X in the appropriate blank to get:

THE TARGET OF THE ?MSN RADIATES.

Although this sentence is the direct output of the second clause, it combines both clauses because it has used the first one to translate the occurrence of ?X in the second. RULETRAN can condense three or more clauses in the same way, if they are chained together by repeated occurrences of variables.

To make the English output seem more natural, the complete translation for a variable, such as THE TARGET OF THE ?MSN for ?X above, is used only once. Any subsequent occurrences of the same variable are translated with a short form, in this example THE TARGET. The short form is also on the property list of the keyword. Short forms have to be chosen carefully to avoid ambiguity. For example, since a mission has both a departure time and a time over target, neither one should have just THE TIME as its short form.

EXPLANATIONS

The User's View

When a rule-based constraint is violated, the initial explanation consists of the name and conclusion of a rule, and it ends with an "Explain" prompt; for example:

BY TARGET-AIRCRAFT-3: THERE IS A SERIOUS CONFLICT
BETWEEN THE
TARGET AND THE AIRCRAFT FOR OCA1001.
Explain?->

A response of YES causes the hypotheses of the rule to be printed.

The hypotheses are displayed in a numbered list. Continuing with the same example, we would get:

1. THE TARGET OF OCA1001 RADIATES
 2. THE RADIATION TIME OF THE TARGET IS RECENT
 3. THE AIRCRAFT OF OCA1001 IS NOT AN F-4G
- Explain?->

By responding with 1, 2 or 3, the correspondingly numbered assertion is expanded with its own list if possible.

In general, assertions in an explanation may or may not be subject to further explanation. Assertions that cannot be explained further are: facts about the current mission parameters, such as the target identity; other facts in the data base, such as the complement of aircraft at the selected airbase; system data such as the time of day; and computational results, such as "2 IS LESS THAN 3"

In the explanations, factual assertions are labeled DATA; computational results are labeled CONDITION. If a factual assertion of either type is selected for expansion, only the message DATA will appear.

Other assertions, which can be explained, are of three kinds: conclusions of inferences, condensations, and INHERITANCE deductions. Conclusions of inferences are recognizable because they begin with BY xxx: where xxx is a rule name. They are expanded recursively.

Condensations are statements formed from two or more clauses or LISP side conditions, as discussed in the previous subsection. They can be expanded into their constituent expressions. For example, the condensation:

1. THE TARGET OF OCA1001 RADIATES

is expanded into:

- 1.1. DATA: THE TARGET OF OCA1001 IS
BE50326-SEARCH-RADAR
- 1.2. INHERITANCE: BE50326-SEARCH-RADAR
RADIATES
Explain?->

INHERITANCE deductions are statements of set membership, set inclusion, or inherited attributed, determined from the logical structure of the data base. An example is assertion 1.2 above. Its expansion is:

- 1.2.1. BE50326-SEARCH-RADAR IS A
PAT-HAND
- 1.2.2. PAT-HAND IS A SEARCH-RADAR
- 1.2.3. SEARCH-RADAR IS A RADAR
- 1.2.4. RADAR RADIATES
Another node?->

Assertions contributing to an INHERITANCE deduction are always facts that cannot be further expanded. The message "Another node?" reminds the user of the opportunity, which could have been exercised at any point, to expand some other part of the overall explanation.

CHOICE GENERATION AND AUTOMATIC PLANNING

From Constraint Checking to Automatic Planning

Initially, KNOBS was a plan checking tool. But the research team quickly realized that a system which can check values filling slots for consistency can be made to automatically select correct values for the slots if only it is given some way to enumerate potential candidates.

The first step along the road to automatic planning was to use some sort of "daemon" (either attached to an \$IF-ADDED facet of a slot or attached as a "daemon" constraint that only has a side-effect) that will compute the values of the slots which can be computed from other slots. Examples of this type of slot are the radio call sign for the mission and the communication frequency. Both can be determined from the "unit" (or "wing") supplying the aircraft.

The second step is to enumerate potential slot fillers in an arbitrary manner. KNOBS can either produce a complete listing of all potential slot fillers, or it can produce a "generator" that, when one value is rejected, produces another potential value. Examples of the kinds of slot values that cannot be enumerated into a complete list are:

- * Numerical values, for example: numbers of aircraft, times of departures.
- * Sets of candidates. KNOBS cleverly avoided ever having to enumerate a set.
- * Other missions or -- more generally -- other plan elements. Examples include refueling missions, orbits for those missions, and various escort missions.

One wants not only a legal plan, but one would like a plan that is, in some sense, "good." The KNOBS researchers knew that global optimization would not be possible (indeed, for the general mission planning problem there are no effective algorithms for achieving global optimization; all of the so-called "optimization" algorithms in fact attempt to find a semi-optimal solution using a fixed set of hard-coded heuristics). Instead, the researchers developed the notion that each set of candidates could be ordered according to some local criteria. Then the first candidate that satisfied all of its constraints would be chosen, and by ordering it would be the "best."

The paragraph above indicates the third step on the way to automated planning: ordering the lists of candidates that can be enumerated in a complete list.

The fourth, and final step, towards automated mission planning is developing a problem-solving strategy. For the KNOBS system, a strategy has a very simple form: it is an ordered list of the slots which are to be filled. KNOBS proceeds as follows: the human planner makes as many of the choices as he wishes. When these choices have been consistently made, KNOBS runs down its ordered list of slots and tries to fill each in turn. If no acceptable candidate for a slot can be

found, then KNOBS "backs up" the list to use some other candidate for a slot earlier on its list. The order of the list turned out to be non-critical. As an experiment, the "correct" order was installed "reversed" (presumably the order opposite to the best was the worst). KNOBS continued to correctly plan missions, but it did an enormous amount of backtracking in the process. However, in absolute terms the problem did not get too bad: the mission automatic planning time did not increase by more than a factor of ten.

KNOBS's Support Facilities

Knobs has five facilities to help the planner make choices for mission items, with increasing levels of initiative exercised by the system. The first is the constraint checking that is performed after the planner proposes a value. The second is the facility for enumerating acceptable choices. A choice is considered "acceptable" if it does not violate any constraint, given the present assignment of item values. After enumeration has been requested, the ordering option, representing the third level of initiative, will list the enumerated choices in order of preference. Fourth, instead of going through the enumeration on each item, the planner can just request "FINISH" to invoke the AUTOPLAN facility; this finds a consistent and preferred value for all of the remaining mission items, or determines that a consistent assignment does not exist. Finally, the system can initiate automatic replanning without an explicit request from the user, when updates to the data base occur that invalidate existing mission parameters.

Does the Air Force want AUTOPLAN?

After all of the experimentation, we still don't know if the Air Force wants, or should want, an AUTOPLAN facility. The arguments against are primarily emotional: "No, we don't want some *** computer making important decisions." There are, however, technical questions that must also be resolved. These questions seem to fall into several camps.

One set of questions has to do with reliability. I find it curious that the reliability of the consistency checks are seldom questioned, but the reliability of the automatic planning -- which is driven by the same consistency checks -- is frequently questioned.

Another set of questions concerns the feasibility of autopanning outside of the stereotypical planning domain. Outside of this domain, the notion of a planning strategy must change from a simple list of slots to something much more complex. The technology for expressing and using these more complex strategies has not been developed.

A third set of questions concerns the flexibility of those plans. If the dozens or hundreds of missions have been planned automatically, then no person really understands them (the understanding, if any, has been captured inside the computer). How can various missions be replanned? If planned by humans, then presumably those humans can remember why choices were made and cleverly replan missions while minimizing perturbations. If the planning was

done by machine, there must be technology that can similarly replan while minimizing the impact of those changes on on-going activity.

ENUMERATION

Note, first of all, that there are two kinds of mission items: those with an explicit list of possible values, and those that take an infinite choice of possible values. The AIRCRAFT item has only the values F-4C, F-111E, etc., and these may be found in the value of the INSTANCES slot of the generic AIRCRAFT frame in the data base. The TD (time of departure) slot, however, can be filled with any date-time combination. Any enumeration of possible values for a continuous-range item is expressed in terms of intervals [A, B] from a low value to a high value. If the value is unrestricted above or below, this is indicated by an arbitrary practical bound; for example, there is an "end-of-time" date about four hundred years from now that serves as an upper bound for all time values.

A simple-minded scheme to generate all acceptable values for an item would be to start with a list or range of all possible values, and then apply each timely constraint to each possible value. We point out first of all that it is impossible to test all the values in a continuous interval, and even for an explicit list this scheme is inefficient.

Instead, KNOBS inverts whichever constraints it can. Inverting a constraint means to find a function that lists all of the values for an item that are acceptable to that constraint, assuming that the other arguments to the constraint are known. A constraint can be inverted, if at all, with respect to any one of its arguments. In general, rule-based constraints cannot be inverted in a practical fashion.

Enumeration, therefore, works like this: the appropriate inversion function of each of the timely constraints is run to get a list or range of values acceptable to that constraint. The set-intersection of these lists is taken to find those values acceptable to all the invertible constraints. Finally, if the result is an explicit list, the timely non-invertible constraints are checked to see if any of the values on that list should be eliminated.

In order to check a rule-based constraint for a proposed value, that value must temporarily be inserted into the item, and removed afterward. If the item had a value to start with, that value is saved before enumeration and restored afterward. There is one more bookkeeping concern: recall (from the section on the non-monotonic logic of constraint checking) that if a constraint fails, all items involved become "trapped"; they must be released when a temporary value is removed.

There are even more bookkeeping details: daemons, whether of the \$IF-ADDED or the constraint-based variety, should not be run when candidates are simply being checked for consideration.

Implementation

In a mission template frame, each item slot has two facets added to support enumeration: \$GENERATORS and \$FILTERS. The \$GENERATORS datum is a list of inverted constraints; the \$FILTERS datum is a list of the remaining constraints that could not be inverted efficiently.

The function ENUMERATE uses the \$GENERATORS and \$FILTERS data to produce an explicit list or range of acceptable values, and places that list on another new facet of the item slot in the mission frame: the \$POSSIBLE facet.

As an example, here is the relevant portion of the UNIT slot in the OCA template:

```
(UNIT
...
($GENERATORS (((UNIT-FROM-P1
  (UNIT-FROM-USERP1)
  (UNIT-FROM-AB-1 (AIRBASE)
    (*AND* AIRBASE))
  (UNIT-FROM-AC-1 (AIRCRAFT)
    (*AND* AIRCRAFT))))))
($FILTERS (((AVAILP1 (TARGET UNIT
  TOT TD AIRBASE AIRCRAFT
  ACNUMBER)
  (*AND*
  (*OPT* TARGET)
  UNIT
  (*OR* TOT TD)
  AIRBASE AIRCRAFT ACNUMBER))))))
...).
```

The generators and filters are expressions that include the arguments of each constraint, and the arguments are modified with timeliness operators in the usual way for constraint references. Filter expressions, in fact, generally just repeat those constraint references in the CONSTRAINTS slot that mention the current item as an argument.

The UNIT item has one filter and four generators. The UNIT-FROM-AB-1 generator, for example, inverts the constraint ABUNITP1, which tests whether a given unit is located at a given airbase. The generator takes the airbase as its argument (it omits UNIT as an argument), and does a data base lookup to list all the units that are located at that airbase; they are found in the UNIT slot of the frame for the airbase.

Discussion of Enumeration

When enumerating candidates to fill some slot, how much effort should go into making sure that the candidate is really feasible? That is, suppose KNOBS was suggesting an airbase from which to fly some mission when all that was really

known was the target and the desired probability of destruction. One might ask that all the candidates satisfy the following increasingly stringent constraints:

1. Candidate be an airbase.
2. Candidate have aircraft that can fly to the target.
3. Candidate have aircraft appropriate for the target (i.e., can fly to the target carrying ordnance good for the target).
4. Candidate have enough aircraft at the appropriate time that can fly to the target with an appropriate ordnance.
5. Candidate have enough acceptable aircraft and if they were used then all other missions could also be planned (i.e., using a resource at the candidate airbase did not lock out a solution to some other mission planning problem).

The last two constraints simply demand that if the candidate is chosen, there exists a complete and acceptable mission plan with that candidate and all already-made choices.

The intention of the KNOBS developers was that enumeration be very simple with little constraint-checking. However, there is nothing to really prevent an exhaustive look-ahead to guarantee (say) condition number 4 above. Indeed, for semi-automated planning such a guarantee might be worth maintaining.

Consider the following alternative mission planner: once a target is selected, a "ghost" mission is autoplaned. This mission is the best consistent with the choices the user has already made (at this point, the user has only chosen the target). As each additional user choice is made, the "ghost" plan is revised or the user is informed that there is no plan consistent with his choices. In such a system, the user might be able to ask for lists of possible candidates for which limited "ghost autopanning" has been performed, or perhaps the only list of possibilities the user can ask for is a list of less desirable candidates which the planning system has not really considered. In the latter case, the only user alternative is to request that some system-selected choice be changed, or some rejected candidate to be explained.

In summary, the enumeration facilities provided in KNOBS merely suggest the types of facilities which could be built using knowledge-based system technology.

ORDERING

An ordering request for a slot ranks the list of acceptable choices, with some values tied. Only some items can be ordered; bounded-range items are not

ordered, and it was also not considered reasonable to order some explicitly listed items.

Choices are ordered on the basis of preference rules; preference rules are like constraint rules except that their conclusions are of the form:

(SUGGEST-item ?MSN type)

where "item" is a mission item like AIRCRAFT or ORDNANCE and "type" is a generic frame having that type of item as instances or individuals. For example, the rule:

```
(AIR-TO-AIR-THREAT
  (((TARGET ?MSN ?TARG))
   ((AIRBORNE-DEFENSE-ACTIVITY ?TARG ?D)
    (IGEQ ?D 4)))
  ((SUGGEST-AIRCRAFT ?MSN FIGHTER)
   (SUGGEST-ORDNANCE ?MSN AIR-TO-AIR)))
```

suggests that fighters with air-to-air ordnance be used when the MIG defense of the target is substantial. When ordering a list of aircraft, therefore, fighters will be given a higher recommendation than others, and when ordering a list of weapon loads, those with air-to-air capability will be preferred.

Discussion of SUGGEST-...

There are several decisions to be discussed concerning the way in which possibilities are ordered by using rules that suggest the possibility have certain features. Before going into the design of the rule-based suggestion mechanism, we should note (as described below) that the ordering is done by an arbitrary LISP function attached to the slot being filled. As has been observed before, KNOBS was an experimental testbed and can thus be excused from not making a strong commitment to doing business in any particular manner. However, the knowledge engineers are unable to depend on the rule-based suggestion mechanisms being activated.

Desirable features are associated with particular frames. That is, when the rule above suggests that the aircraft of a mission be a "fighter," that suggestion is that the a/c be an instance of the frame FIGHTER. The problem with this decision is that the FRL frame system enforces a strict AKO tree. Thus if we wanted a rule to recommend that some aircraft have both the features "all-weather" and "fighter" we would need to insure either that "fighter" was a subclass of "all-weather" or vice versa. However, not all fighters are all-weather, and there are some aircraft that are all-weather but are not fighters, so neither can be a subclass of the other.

There are other techniques that could have been used to achieve a similar end. For example, the suggestion rules could have given properties that were desirable for a candidate to have. The candidates could have inherited sets of properties. The ordering function could then have scored each candidate by the number of desirable properties it had. The reason a more sophisticated

suggestion mechanism was not implemented is simply that the existing one was perfectly adequate.

Another weakness (which was similarly left unaddressed because more sophistication was not required) is that the suggestion did not have a measure of strength associated with it. Thus if two candidates were preferred for different reasons, there was no way to judge between the two based on evidence from the suggestion rules.

Implementation

The ordering procedure for a mission item is kept in the \$ORDER facet of its slot in the mission template. The ordering function calls a version of INFER with a clause of the form (SUGGEST-item mission ?type). For example, to get aircraft recommendations for mission OCA1001, the clause to be inferred would be (SUGGEST-AIRCRAFT OCA1001 ?AC-TYP"). The actual function called is INFER-ALL, which differs from INFER in that it returns a list of all possible bindings (and associated audit trails) for the conclusion variable, instead of just the first one it finds.

The responses from INFER-ALL are then used to order the list in the \$POSSIBLE facet, and the new ordered list is displayed for the planner. Meanwhile, the list of responses from INFER-ALL is placed in the \$RECOMMEND facet.

Explanation

Once the item has been ordered, it is possible for the user to request an explanation. The function that handles these requests is kept in the \$EXPLAIN facet of the slot of the mission template.

To explain a value on the ordered list, the \$EXPLAIN function looks at each response on the INFER output in the \$RECOMMEND facet. For each type recommended, it is determined whether or not the given value is of that type. The displayed message includes that fact, the probability of destruction associated with that value if relevant, and the conclusion of the rule, obtained from the audit trail.

AUTOPLAN

The FINISH or AUTOPLAN request asks the system to take over the task of filling in all remaining mission items. AUTOPLAN has been implemented, so far, only for OCA missions without refueling. It can be called at any time after the target and PD have been specified.

The AUTOPLAN facility simulates the role of a user who simply accepts the recommendations made by the system. It constructs a plan by going through the mission items in a fixed order, selecting the highest-ranked value generated by the ORDER function for each item, if ordering is possible. Otherwise it takes the first enumerated value. If the enumerated values are in a continuous range, it selects the least (earliest, in the case of a time) value in that range.

That would be all there was to the program if such an algorithm were guaranteed to find a solution. It is not, even in cases where a consistent solution exists. It might, in fact, drive us to a cul-de-sac, similar to the one in the sample planning session, where KNOBS said that there were no choices of airbase consistent with previous selections of target (BE70501-RUNWAY) and aircraft (F-111E). But, that did not mean we could not plan a strike against the target. It did mean that we would have to change our choice of aircraft before continuing. The failure could not be predicted before a choice of airbase was attempted, because constraints can't be checked before their arguments are available.

Backtracking

When AUTOPLAN hits a cul-de-sac, because enumeration fails to yield any acceptable values, it backtracks to a previous item and replaces it with the next acceptable value in the ordered list. If the item is enumerated with a range of values, some fixed increment is added to the old value to get a new candidate. In the case of a time, for example, five minutes is added. (KNOBS didn't really try to be clever in enumerating continuous values).

If all acceptable values have been exhausted, backtracking continues to a previous item. If there are no previous items except the ones considered to be fixed because they were specified by the planner, AUTOPLAN gives up.

AUTOPLAN is smart enough not to try changing a previous item when it won't do any good - because it was not involved in any constraint that failed. To do this, AUTOPLAN uses a version of ENUMERATE that keeps track of which constraint failed for each rejected value. That way, when ENUMERATE reports that there are no candidates, AUTOPLAN can look at the arguments of the failed constraints and see what the latest item was that might fruitfully be changed.

The order in which items are chosen for filling makes a significant difference to the speed of the search. For example, choosing the unit before the airbase will cause a lot of unnecessary backtracking, since the constraint that the airbase must be within range of the target will cause many units to fail when an airbase is chosen, whereas if the airbase is chosen first, most of the units would be eliminated without backtracking. The order used by AUTOPLAN for OCA missions is the following: AIRCRAFT, AIRBASE, ORDNANCE, ACNUMBER, UNIT, TOT, TD. It has not been proved that this order is optimal, but a combination of common sense and observation of AUTOPLAN traces suggests that it is reasonable.

Example

Here is an example of AUTOPLAN at work in a situation that required some backtracking. We will follow the trace that is normally displayed by AUTOPLAN.

The mission starts out with a required PD (probability of destruction) of .95 for BE70501-RUNWAY.

Ordering AIRCRAFT

Candidates are F-5E F-15 F-4G

F-111F F-4E F-4C F-4D

F-111E A-10

Trying F-5E for AIRCRAFT

Enumerating AIRBASE

Enumeration failed - no candidates.

What happened? The program discovered (though it did not say so) that the only airbase with F-5E's is Spangdahlem. But Spangdahlem turned out to be too far from the target for an F-5E. The trace continues:

Conflict with AIRCRAFT Trying F-15 for AIRCRAFT Enumerating AIRBASE
Candidates are SPANGDAHLEM RAMSTEIN HAHN BITBURG Trying
SPANGDAHLEM for AIRBASE Ordering ORDNANCE Enumeration failed -
no candidates. Conflict with AIRCRAFT.

The problem was that while the data base lists OCA among the roles for an F-15, the only ordnances provided are for air-to-air combat. The program has noted that the conflict is with AIRCRAFT, so it will not bother to try other airbases until after the aircraft is changed to the next one in line.

Trying F-4G for AIRCRAFT

Enumerating AIRBASE

Candidates are RAMSTEIN

The set of candidates for airbase is now different because there is a constraint that checks that the chosen aircraft, now changed, is available there.

Trying RAMSTEIN for AIRBASE

Ordering ORDNANCE

Enumeration failed - no candidates.

Conflict with AIRCRAFT

A similar tale. The F-4G does not have ordnance suitable for runways. Another aircraft must be tried.

Trying F-111F for AIRCRAFT
 Enumerating AIRBASE
 Candidates are Wiesbaden
 Trying Wiesbaden for AIRBASE
 Ordering ORDNANCE
 Candidates are A10 A1 A6 A9
 A8 A4 A2 A7 A14 A5 A3
 Trying A10 for ORDNANCE
 Ordering ACNUMBER
 Candidates are 3 4
 Trying 3 for ACNUMBER
 Enumerating UNIT
 Candidates are 113TFW
 Trying 113TFW
 Enumerating TD
 Candidates are 3-Sep-82 14:11:50 28-Jun-2341
 23:59:59

The candidates for TD are times in a range with the given lower and upper bounds. The upper bound here is the default date far in the future that is used when no other bound can be calculated.

Trying 3-Sep-82 14:11:50 for TD
 Enumerating TOT
 Candidates are 3-Sep-82 14:37:50
 3-Sep-82 14:48:04
 Trying 3-Sep-82 14:37:56 for TOT
 Plan completed.

AUTOMATIC REPLANNING

In an operational setting, various events may occur that force changes in the KNOBS data base; and these changes, in turn, can affect missions currently being planned. Such events may include changes in the weather, the closing of an airfield, or orders that alter preference or constraint rules.

It was mentioned in the section on the English subsystem that statements that update the data base, such as "RAMSTEIN IS DOWN", are recognized by production rules. The immediate action is to change the value under the OPERATIONAL slot of the Ramstein frame from true to false. But quite a bit more happens. The message displayed by KNOBS is:

Airbase RAMSTEIN is down!
 Mission MSN1002 depends on it.
 Replacing airbase RAMSTEIN of OCA1002 with
 SPANGDAHLEM
 Replacing UNIT 116TFW of OCA1002 with 120TFW

The Forward-Chaining Rule Interpreter

This activity was generated by the forward-chaining rule interpreter. When the English subsystem changed the OPERATIONAL status of Ramstein, that action

triggered a call on ASSERT, the main function of the interpreter, which established a hypothesis in a forward-chaining rule.

The triggering was accomplished by an \$IF-ADDED facet in the OPERATIONAL slot, not in the RAMSTEIN frame, but in the generic AIRBASE frame. The actual entry there is:

```
($IF-ADDED
  ((ASSERT
    (LIST (LIST (COND (:VALUE 'AB-UP)
      (T 'AB-DOWN)))
    :FRAME))))
```

The \$IF-ADDED procedure is evaluated in a context where :FRAME is the current mission frame and :VALUE is the value that has just been added to the current slot. If the value is NIL, i.e., false, the condition AB-DOWN is asserted for the airbase of the current mission.

The rules involved are found by looking on the REFERENCED-BY property of AB-DOWN; this leads us to rule AB-DOWN1:

```
(AB-DOWN1 ((?AB-DOWN ?AB))
  ((IS-A ?MSN OCA))
  ((AIRBASE ?MSN ?AB)))
  ((MSG Airbase (?AB) is down!
    Mission (?MSN) depends on it. :))
  (ASSERT (AB-DOWN ?AB))
  (ASSERT (USED-AB-DOWN ?AB ?MSN)))
```

The initial call on ASSERT established the AB-DOWN hypothesis and bound ?AB to RAMSTEIN. Like INFER, ASSERT attempts to establish the remaining hypotheses with successful bindings for their variables. It will find each of the OCA missions using RAMSTEIN, and, for each one, take the three actions listed in the conclusion part of the rule.

The actions in forward-chaining rules are of a few standard types. The most important ones are ASSERT, MSG, and REPLACE-VALUE. ASSERT triggers other rules; MSG prints a message; and REPLACE-VALUE updates a slot. The ASSERT action on AB-DOWN, above, reactivates AB-DOWN1 so that its consequences can be obtained for other missions from the same airbase.

In this example, the condition USED-AB-DOWN was asserted for RAMSTEIN in the current mission, triggering a second rule which checked whether KNOBS should attempt an automatic replacement; that one asserted a NEEDS-REPLACING condition for the AIRBASE of the current mission, triggering the AB-REPLACEMENT rule:

```

(AB-REPLACEMENT
(((NEEDS-REPLACING AIRBASE ?MSN))
((AIRBASE ?MSN ?OLDAB))
((COMPATIBLE ?MSN AIRBASE ?NEWAB)
 (NEQ ?OLDAB ?NEWAB)))
((REPLACE-VALUE ?MSN
      AIRBASE
      ?NEWAB)
 (MSG Replacing (?OLDAB) of (?MSN)
  with (?NEWAB):)
 (ASSERT (NEEDS-REPLACING UNIT
      ?MSN))))

```

The COMPATIBLE keyword is special in forward-chaining rules. It is the crux of the replanning activity. It invokes ENUMERATE to find a binding for the indicated item of the given mission that is consistent with all constraints. If ENUMERATE fails to find any, the current action is to revert to manual replanning for the affected mission. One might think of calling AUTOPLAN here, but all of the mission items have already been given values; the planner has to decide which ones to cancel.

Limitations of Replanning

After the planning activity was complete, KNOBS lost track of who (mission planner or AUTOPLAN) filled the various slots. Thus KNOBS could not retract its own choices in the example given above. More seriously, the user cannot usually change values in slots after AUTOPLANNING because KNOBS complains that the choice is inconsistent with previously made choices. Even if one wanted to "trap" the choices made in AUTOPLANNING, one couldn't because that information is not preserved.

The paragraph above illustrates the tip of the iceberg: in order to effectively replan, the reason choices were made must be maintained. While it is difficult to get reasons from the user (who probably doesn't have any he is willing to give), the reasons for making some selection during AUTOPLAN are certainly known to the system. Even for some user choices, reasons can be hypothesized -- particularly if the user has asked the system some pointed questions prior to making a choice.

CONCLUSIONS

KNOBS's frame-based instantiation mechanism is an interface between declarative knowledge and procedural activity. It is a representation of a constraint satisfaction problem, but organized as an inference machine. The system is driven by input data which traditionally has come directly or indirectly from a user's choice, but, in other applications, may come from external system sensors. Whatever the source, the inputs drive constraints to test their validity, and daemons to take appropriate responsive actions. These constraints and daemons access the rule and frame knowledge bases, and their effect is the construction or modification of a frame.

KNOBS currently represents goals only implicitly through constraint satisfaction.

The AUTOPLAN facility is in fact a back-chaining engine based on this limited form of goal representation. The lack of a more abstract goal representation is what keeps KNOBS from being a full-fledged planning agent, and the next generation of the system is committed to development in this direction.

Nevertheless, the appropriateness of this approach has been demonstrated in some markedly varied problem domains. The common thread is that this architecture is applicable to any situation where a "planning assistant" or "advice giver" is useful, where action in a given context requires a critical response. In spinoff efforts on other applications, the KNOBS problem representation has evolved from representing plans among cleanly packaged quantities (aircraft wings and standard configuration loads) in tactical mission planning, through a multidimensional balance of more dynamic objects (ship groupings) in naval mission planning, through scheduling problems in crew activity planning for NASA, to system monitoring in a propellant loading application for NASA.

Features in KNOBS

KNOBS worked with a realistically sized database. It combined problem-solving AI mechanisms, a natural language interface, and a "full sized" knowledge base into a single system. As a very early attempt to build a full-scale knowledge-based system, KNOBS was a pioneering research project.

Being implemented on a computer in DEC's PDP-10 family, KNOBS suffered from having only 18 bits of address in a normal machine pointer: this is a relatively small address space, although standard at the time KNOBS was started. In order to get around this hardware limitation, a *frame swapper* was implemented. Although necessary, this tended to cause a lot of paging activity on time-shared systems, which in turn meant that KNOBS was slow for reasons unrelated to the efficiency of the algorithms used by KNOBS.

It seems that most computer-aided planning systems have either left the planning entirely to the user (supplying graphic output, and some tracking of resources), or has completely automated the planning process. KNOBS stood almost alone in recognizing that planning must be an interactive process, in which the initiative for making decisions is sometimes taken by the human user

and at other times taken by the computer system. This *mixed initiative* approach to planning appears to be the system feature that forces the use of AI techniques.

Historically, the *mixed initiative* interaction that KNOBS supports has changed the way man-computer interactions are viewed within some parts of the Air Force. The critical contribution was not so much that *mixed initiative* was desirable, but that *mixed initiative* could be obtained using currently available technology.

KNOBS emphasized *natural language* interaction throughout, beginning with a syntactic pattern matcher in MICROKNOBS to a script-based dialogue system in KNOBS's successor KRS. Great insight into the demands that would be placed on the natural language system were obtained. In particular, we found some indications that the natural language understanding problem as it relates to a knowledge-based system "front end" is significantly simpler than, for example, the *story understanding* problem.

KNOBS was an experimental vehicle. As such, it is not surprising to find that there are several distinct and separate mechanisms implementing the same conceptual capability. One example of this is the duplication occurring between *constraints*, *rules*, and the *FRL daemons* attached to \$IF-ADDED and \$IF-NEEDED facets of a slot. As time went on, however, these multiple mechanisms led to an increasingly unwieldy implementation: developers and knowledge engineers could never depend on KNOBS to uniformly access and use information.

Another area in which KNOBS was an early pioneer was in multi-directional inheritance. The notion explored is not that of "class mixing," but of viewing other ordered relations -- "a part of", "a member of", "located at", etc. -- as being much like "a kind of."

Critical Decisions and Insights

The KNOBS effort recognized that stereotypical planning is a far simpler type of planning than usually discussed in the artificial intelligence literature. The people behind KNOBS realized that stereotypical planning problems could be solved relatively easily by a generate-and-test paradigm. While it is commonly thought that this paradigm does not lead to a computationally tractable solution in many applications, it does lead to one for stereotypical planning, as KNOBS showed. Furthermore, a number of applications appear to be almost entirely examples of stereotypical planning.

The KNOBS effort recognized that *constraints* should be handled specially (not just another form of rule) for reasons of both efficiency and conceptual clarity. This insight continues to be a major guide in ongoing MITRE efforts.

The KNOBS effort explored the notion of a *constraint timeliness calculus*. Indeed, some sort of technique that separates "what should be done" from descriptions of "when" and "how" seems to be important, both from efficiency

and conceptual points of view. The KNOBS implementation of these notions appears, in retrospect, to have been somewhat of an overkill.

Having decided that *constraints* could serve as the basis for detecting inconsistencies (and, later, for automatically formulating consistent plans), the KNOBS designers recognized that the rule interpreter needed only to support backward chaining.

The KNOBS effort was aided by using an existing body of code -- FRL developed at MIT's AI Lab -- to implement the notions of *frames*. However, as we have shown, the benefit may have turned unexpectedly into a deficiency.

KNOBS and its predecessor MICROKNOBS both emphasized the advantages to be obtained by keeping track of the justifications for decisions and suggestions. While these advantages are recognized today, they were somewhat new when MICROKNOBS was developed. Even though they are well recognized, today's implementations of "expert" systems do not generally track dependencies to the degree that KNOBS did.

KNOBS used natural language both for input and output. While there is some evidence from KNOBS's successors that menu-driven approach is adequate for user input, KNOBS clearly showed that English explanations are essential. KNOBS was successful in generating these English explanations and descriptions from its internal LISP-like representations.

KNOBS emphasized the advantages in allowing rules and constraints to be changed by the user. While the *authority* to make changes in a system's knowledge base should be carefully controlled, the *capability* to make changes must be preserved. KNOBS was one of the first systems to demonstrate the ability to change rules through its natural language interface.

Weaknesses and Regrets

KNOBS made little commitment to standardized control flow, to organizational and architectural conventions, or to the principle *one fact = one declaration*. This is understandable, given that KNOBS was an experimental testbed. However, KNOBS paid a high price for this flexibility: it cannot be said that KNOBS embodied a theory of stereotypical planning; rather, parts of this theory are revealed by the results of the KNOBS project.

The history of the KNOBS implementation effort contains an important general lesson: in any system with inference mechanisms in which hand-coded programs can also be attached to the objects being manipulated by the rules, self-preservation will encourage the implementors to write code rather than add "rules." The end result of this will be a system where much of the knowledge is encrypted in code.

KNOBS use of the FRL implementation of the notion of *frames* turned out to be somewhat of an error. This implementation pays a high price in terms of performance for its generality. For example, when one frame points to another, it usually does so by giving the *name* of the frame in the form of a LISP atom. To find the actual frame structure thus involves a search of that atom's property

list: linear searches are slow. Similarly, in order to dynamically add new slots to a frame, FRL kept the slots in an association list: again a linear search was required to find the value in a slot. However, KNOBS's only use of that ability was to show in a demonstration that one could dynamically add new slots (but the system was never able to use that new information): clearly the performance price was too high. Another source of performance degradation centers on inheritance: doing inheritance on each access (rather than doing all inheritance on instantiation) is appropriate only when the information being inherited is changing rapidly. KNOBS did not inherit dynamic data.

KNOBS made a decision to view a slot with several entities filling it as a "multiple-valued" slot rather than as a "set-valued" slot. We found, as a result of experimentation, that the "set-valued" interpretation was more productive: for example, one might like to constrain the size of the set to be "less than 5." Such a constraint cannot be written if the "multiple-valued" interpretation is used.

KNOBS uses NIL to indicate failure to prove something and to indicate that something is false. This interpretation of *failure as negation* is an error, as described earlier. KNOBS is not alone in making this mistake: we continue to see this error in newly developed systems.

KNOBS explored the premise that considerations of the interface should be deeply embedded into the structure of the inference mechanisms. This decision led to a number of unfortunate consequences, including an inability to modify the way the inference mechanism functioned without simultaneously reorganizing the interface's operation. Of more conceptual significance, this led to poor choices when the needs of automatic planning conflicted with the needs of the interface.

KNOBS used *conceptual dependencies* as a representation scheme in its natural language interface. We have come to regard this scheme as *implementationally dangerous* for several reasons, including the comparatively large number of arbitrary decisions required of the implementor.

AD-A174 375

KNOBS (KNOWLEDGE-BASED SYSTEM) - THE FINAL REPORT
(1982)(U) MITRE CORP BEDFORD MA R H BROWN ET AL
AUG 86 RADC-TR-86-95 F19628-79-C-0001

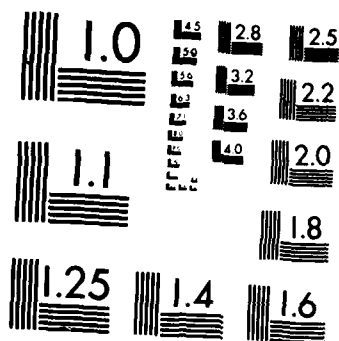
2/2

UNCLASSIFIED

F/G 5/2

NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

APPENDIX

A SAMPLE PLANNING SESSION

In a certain saved LISP environment is a function "KNOBS." When executed, it places the user in the KNOBS-level command environment. On a display terminal, the menu in Figure 17 will appear. (This display is from a DEC VT100; displays for other terminals will vary somewhat. An interface with equivalent functionality is available for printing terminals.)

KNOBS

(KNOBS) 12-Oct-82 23:33:29 Current time ZULU = 14-Oct-82 12:32:31

PLAN

REPLAN

DELETE

CHANGE-DISPLAY

EXIT

Existing Missions:

Mission	Type	Target	TOT	A/C	#	SCL	Refuel?	TD	Items not defi
OCA1000	OCA	BE50326	1610	F-4C	4	B2	****	1542	

Use function key f5 for help.

Figure 17. KNOBS top-level Menu Display

Five command selections, from PLAN to EXIT, are shown. Below them is a summary of missions whose planning is in progress or completed during this terminal session; ordinarily this list would be empty when KNOBS is first entered, but one is shown to illustrate the format.

The display cursor is initially on PLAN, and will skip from selection to selection in that area under control of the terminal's cursor movement keys. A carriage return selects the item at the cursor.

Suppose PLAN is selected. Below the menu, the user is asked for a mission type, one of OCA (Offensive Counter-Air), RFL (Refueling), or a few others. The answer is typed in: say, OCA. A new menu appears, shown in Figure 18. This actually represents a mission frame, and the menu items are the mission items to be filled in. As in the KNOBS-level display, a carriage return selects the item beside the cursor, and further interaction takes place in the text area below the menu.

This is OCA1001 (a ground-strike offensive counter-air mission)

The calculated probability of destruction cannot be computed.

TARGET	PD	AIRCRAFT
AIRBASE	ORDNANCE	ACNUMBER
UNIT	TD	TOT
REFUELSVC -optional-	CALL-SIGN -auto-	TRANSPONDER -auto-

Use function key f5 for help.

Figure 18. Mission-level Menu Display

Instead of selecting a mission item, the user could also enter one of several generally available commands. For example, there is a "help" command that gives the user general instructions on what to do, and an "English" command that puts the user in an environment that responds to English statements and questions. Commands may be entered either with function keys (depending on the terminal type) or with one- or two- character designators like *H and *E.

Suppose that the TARGET item is selected. The system prompts for a target; it may be entered either by name or by its Bombing Encyclopedia code. A question mark "?" at this point would cause the user to be led interactively through the tree of all possible targets.

Rather than give a particular value for an item, the user can restrict its allowable values with an expression of the form:

RESTRICT x (TO/THROUGH) y

or

RESTRICT x y z ...

This creates a new constraint on the values for that item. Restrictions are useful primarily to constrain automatic planning for the affected items. A restricted item is marked on the screen by an asterisk. Typing a question mark at the item will describe the restriction in addition to the information normally given. To remove a restriction, the user types UNRESTRICT.

Let us assume that the values below have been entered. PD is a design parameter, the planner's statement of the minimum acceptable probability of destruction, and ACNUMBER is the number of aircraft assigned to the mission.

TARGET: BE70501-RUNWAY
PD: .85
AIRCRAFT: F-111E
ACNUMBER: 3

In choosing an AIRBASE, now that some other items have already been chosen, it makes sense to list only those airbases that are consistent with the chosen items. This is the purpose of the enumeration facility, which is requested by a function key or "#". The result is the following message:

There are no choices consistent with previous decisions.

The AIRBASEs BITBURG, HAHN, MILDENHALL, RAMSTEIN, SEMBACH, SPANGDAHLEM, and WIESBADEN are not acceptable because the AIRBASE does not have complements of F-111E aircraft.

The AIRBASE ALCONBURY is not acceptable because the target BE70501-RUNWAY is not within range of the airbase using F-111E aircraft (without refueling).

There is a world of difference between this explanation and the computer having told the planner nothing more than "TILT, YOU LOSE!". Our way, the planner understands the problem that has to be solved, and can start applying his efforts in that direction.

In this example, no consistent choice was available. In a deeper sense, one could certainly say that the previous values for the other items were already inconsistent. But the computational burden of always probing for this possibility would generally be prohibitive.

An obvious feature of the above explanation is that different airbases may be excluded for different reasons. Another, perhaps less obvious, feature is that this explanation also follows the principle that one does not bother explaining details if there is a higher, overriding consideration. In this case, the airbase Mildenhall, like Alconbury, is in England and would be too far from the target, even if it had the F-111E's. But why talk about that when it doesn't?

Let us go ahead with Hahn, with the intention of substituting another aircraft next. Insertion of Hahn for the AIRBASE item causes the values for the AIRCRAFT and AIRBASE to be highlighted in reverse video on the display, indicating a conflict. This is accompanied in the text area by the reminder:

The AIRBASE HAHN does not have complements of F-111E AIRCRAFT.

Which aircraft are there at Hahn? One way to find out such domain-specific information is to use a function key or "*E" to request the English interface. The prompt "English:" appears on the newly cleared display. Now we can

simply type: "WHICH AIRCRAFT ARE THERE AT HAHN?". The response is:

HAHN has F-4C's, C-130's, RF-4C's and F-15's.

The mission menu is restored after an "OK" to the English prompt. If F-4C is entered now, it lights up, along with ACNUMBER, and there is a message noting a conflict. If "?" is entered now, the following message appears on a cleared display:

The choice for AIRCRAFT is in question because:

BY AIRCRAFT-ACNUMBER-1: THERE IS A MODERATE
CONFLICT BETWEEN
THE AIRCRAFT AND THE NUMBER OF AIRCRAFT FOR
OCA1001
Explain?->

A user response of "Y" or "YES" results in:

1. THE AIRCRAFT OF OCA1001 IS ONE OF F-4C F-4D
 2. THE NUMBER OF AIRCRAFT OF OCA1001 IS ODD
- Explain?->

The numbered facts are the hypotheses of the rule AIRCRAFT-ACNUMBER-1 interpreted for the current mission; the stated conflict is the conclusion of that rule. This rule derives from an Air Force training convention in which pilots are instructed to fly F-4's in flights of two and four in particular formations, with each aircraft having a well-defined role.

This seems clear enough; the next step is to change the ACNUMBER to 4 and go on to assign a weapon load, at the ORDNANCE item. Enumerating the ORDNANCE yields a message stating that there are six suitable values, and offering to order them. If ordering is not requested, the following menu appears:

Select a value for ORDNANCE
-ABORT- -ORDER- B2 B4 B5 B6 B17 B19

The "ABORT" selection, in this and other menus, simply returns to the previous menu. If ordering is selected, either initially or from the above menu, a different menu appears:

Select a value for ORDNANCE

-ABORT-
B5
B4 B2 B6
B17
B19

The three on the second line of this list are tied. Why was B5 considered best? Move the cursor to B5 and hit "?". The answer is:

The standard configuration load B5 contains:

11 M-117
1 ECMPOD
2 AIM-7E

B5 CONTAINS ECM

BY SAM-ORDNANCE: ECM ARE RECOMMENDED FOR
OCA1001

THE SINGLE-PLANE PROBABILITY OF DESTROYING A
ENEMY-AIRBASE-RUNWAY WITH B5 IS .5

Hit the space bar to return to the menu...

The rule SAM-ORDNANCE recommends ECM (Electronic Counter-Measures), and B5 has it. It was ranked over other loads with ECM because it had a higher probability of destruction for this target. If we want to know why ECM is recommended, one way to find out is to request PRINT SAM-ORDNANCE in English mode.

SAM-ORDNANCE

IF:

1: THERE IS A SAM WHICH DEFENDS THE
TARGET OF A ?MSN

THEN:

1: ECM ARE RECOMMENDED FOR THE ?MSN

The remaining items for the user to fill in are UNIT, the tactical fighter wing or squadron number; TD, time of departure; and TOT, time over target. KNOBS can choose values for these automatically by trying the ordered list of candidates for each item, from the top down, searching for some combination which has no conflicts. This automatic planning service is requested from the command menu, obtained by a function key or "*C". The command menu is placed below the mission display and looks like this:

Select a control command

-ABORT- BUGOUT DONE SWITCH
FINISH [AUTOPLAN]

If FINISH is selected, the plan is completed, printing a trace of the search as it proceeds:

Initializing automatic planning.
Ordering UNIT slot
Candidates are (109TFW 110TFW 111TFW 112TFW
108TFW)
Trying 109TFW in UNIT slot
Enumerating TD slot
Candidates are (14-Oct-82 18:14:35 24-Aug-2341 23:59:59)
Trying 14-Oct-82 18:14:35 in TD slot
Enumerating TOT slot
Candidates are (14-Oct-82 18:45:41 14-Oct-82 18:55:49)
Trying 14-Oct-82 18:45:41 in TOT slot
Plan completed

This search was uncomplicated because no conflicts were found and hence no backtracking was necessary. The trace illustrates what the enumeration and ordering facility does with continuous-valued items such as times. A range is given. The lower end of the TD range is the earliest time the aircraft could be prepared for takeoff, starting at the present time. The upper end is an arbitrary figure far in the future. The TOT also has a range, determined by the TD, the flying time to the target, and the maximum flying time of the aircraft.

The planner is now back at the mission menu, where items can be changed if desired. If it is satisfactory, the planner returns to the control menu, and selects DONE. KNOBS finishes up the plan with some items (marked -auto-) that are calculated automatically, such as transponder frequency. Finally, it checks the target area for nearby SAM's, and prints a warning message if there are any:

****WARNING****

A SAM threat exists for this mission due to the following
SAMs
BE50316 KOTHEN-B, an SA-4

Returning to the KNOBS menu, the planner can begin another mission, replan or delete the one just finished, or exit KNOBS.

BIBLIOGRAPHY

- [AIKENS 80] Aikens, J. S., "Representation of Control Knowledge in Expert Systems," Proceedings First Annual National Conference Artificial Intelligence, Stanford University, August 1980, pp. 121-123.
- [BIRN 81] Birnbaum, L., and Selfridge, M., "Conceptual Analysis," in *Inside Artificial Intelligence: Five Programs Plus Miniatures*, Schank, R., Riesbeck, C. K. (eds), Lawrence Erlbaum Associates, Hillsdale, NJ, 1981.
- [BOBROW 77] Bobrow, D. G., Kaplan, R., Kay, M., Norman, D., Thompson, H. S., and Winograd, T., "Gus, a Frame-driven Dialog System," *Artificial Intelligence*, April 1977, 8 (2).
- [CALL 82] Callero, M., Jamison, L., Waterman, D. A., "TATR: An Expert Aid for Tactical Air Targeting," N-1796-ARPA, The Rand Corporation, January 1982.
- [CLME 81] Clocksin, W. F. and Mellish, C. S., *Programming in Prolog*, Springer-Verlag, New York, 1981.
- [CULLINGFORD 78] Cullingford, R., "Script Application: Computer Understanding of Newspaper Stories," Research Report 116, Department of Computer Science, Yale University, 1978.
- [CULLINGFORD 82] Cullingford, R. and Pazzani, M., "Word Meaning Selection in Multimodule Language-Processing Systems," TR-82-13, EE&CS Dept., University of Connecticut, 1982.
- [DEJONG 79] DeJong, G., "Skimming Stories in Real Time: An Experiment in Integrated Understanding," Research Report 158, Department of Computer Science, Yale University, 1979.
- [DOCMC 78] J. Doyle and D. McDermott, "Non-Monotonic Logic I," AI Memo 486, MIT Artificial Intelligence Laboratory, Cambridge, Mass., August 1978.
- [ENGELMAN 79] Engelman, C., Berg, C. H. and Bischoff, M., "KNOBS: An Experimental Knowledge Based Tactical Air Mission Planning System and a Rule Based Aircraft Identification Simulation Facility," Proc. Sixth Inter. Joint Conf. Artificial Intelligence, Tokyo, 1979, pp. 247-249.
- [ENGELMAN 80] Engelman, C., Scarl, E. A., and Berg, C. H., "Interactive Frame Instantiation," Proceedings of the First Annual Conference on Artificial Intelligence, Stanford, 1980, pp. 184-186.
- [ENST 81] Engelman, C. and Stanton, W. M., "An Integrated Frame/Rule Architecture," Proceedings of NATO Symposium in Human and Artificial Intelligence, Lyon, October, 1981.

[ERIC 79] Ericson, L. W., "Translation of Programs from MACLISP to INTERLISP," MTR-3874, The MITRE Corporation, Bedford, MA, November 1979.

[FAIN 81] Fain, J., Gorlin, D., Hayes-Roth, F., Rosenschein, S., Sowizral, H., and Waterman, D., "The ROSIE Language Reference Manual," N-1647-ARPA, The Rand Corporation, December 1981.

[FEIG 71] Feigenbaum, E. A., Buchanan, B. G., and Lederberg, J. "On generality and problem solving: A case study involving the DENDRAL program," In *Machine Intelligence 6*, eds. B. Meltzer and D. Michie. New York: American Elsevier, 1971, pp. 165-190.

[GOLDMAN 75] Goldman, N., "Conceptual Generation," in *Conceptual Information Processing*, Schank, R. (ed), Ninth-Holland Publishing Company, 1975.

[GARVEY 78] Garvey, T. D., Fischler, M. A., Martin, R. A., and Sinko, J. W., "Machine Intelligence Based Multi-Sensor ESM System," Interim Technical Report, SRI International, Menlo Park, July 1978.

[GROSZ 77] Grosz, B., "The Representation and Use of Focus in Dialog Understanding," Ph.D. Dissertation, University of California at Berkeley, 1977.

[HEND 78] Hendrix, G. G., Sacerdoti, E. D., Segalowicz, D., Slocum, J., "Developing a Natural Language Interface to Complex Data," *Association for Computing Machinery Transactions on Database Systems*, Vol. 3, No. 2, June 1978.

[KATZ 63] Katz, J. S. and Fodor, J. A., "The Structure of Semantic Theory," *Language*, 39, 1963.

[LEHNERT 78] Lehnert, W., "The Process of Question Answering, A Computer Simulation of Cognition," Lawrence Erlbaum Associates, Inc., 1978.

[LEHNERT 79] Lehnert, W., "Representing Physical Objects in Memory," Research Report 1978, Department of Computer Science, Yale University, 1979.

[MCDE 80] McDermott, D., "Non-Monotonic Logic II," Research Report 174, Yale University Department of Computer Science, New Haven, Conn., February 1980.

[MINS 75] Minsky, M., "A Framework for Representing Knowledge," in *Psychology of Computer Vision*, ed. Patrick H. Winston, McGraw-Hill Book Co., New York, 1975.

[MOGL 83] Mogilensky, J., Dalton, R., Scarl, E., "Manned Spaceflight Activity Planning with Knowledge Based Systems," presented in AAAI-83 Computers in Aerospace Conference, Hartford, CT, October 1983.

[PAZZANI 83] Pazzani, M. J., and Engelman, C., "Knowledge Based Question Answering," Proceedings of the Conference on Applied Natural Language Processing, Santa Monica, February 1983.

[PAZZANI 83] Pazzani, M. J., "Interactive Script Instantiation," Proceedings of the Third Annual National Conference on Artificial Intelligence, Washington, D.C., August 1983.

[PRETTY 78] Pretty, R. T., (Ed.), "Jane's Weapon Systems 1978," Franklin Watts Inc., New York, 1977.

[REICHMAN 81] Reichman, R., "Plain Speaking: A Theory and Grammar of Spontaneous Discourse," Ph.D. Dissertation, Harvard University, 1981.

[REISBECK 76] Reisbeck, C., and Schank, R. C., "Comprehension by Computer: Expectation Based Analysis of Sentences in Context," Research Report #78, Department of Computer Science Yale University, 1976.

[REIT 79] Reiter, "A Logic for Default Reasoning," Technical Report 79-8, University of British Columbia Department of Computer Science, Vancouver, B.C., July 1979.

[ROBINSON] Robinson, J. A. and Sibert, E. E., "The LOGLISP User's Manual," School of Computer and Information Science, Syracuse University December 1981.

[ROGO 77] Roberts, R. B. and Goldstein, I. P., "The FRL Manual," MIT AI Lab., Memo 409, September 1977.

[ROBERTS 77] Roberts, R. Bruce, and Goldstein, Ira P., "The FRL Primer," MIT AI Lab. Memo 408, July 1977.

[ROSENSCHEIN 75] Rosenschein, S. J., "Structuring a Pattern Space, with Applications to Lexical Information and Event Interpretation," Doctoral Dissertation, University of Pennsylvania, Philadelphia, 1975.

[SCHANK 72] Schank, R. "Conceptual Dependency: A Theory of Natural Language Understanding," Cognitive Psychology, Vol. 3, No. 4, 1972.

[SCHANK 77] Schank, R. and Abelson, H., *Scripts, Plans, Goals, and Understanding*, Lawrence Erlbaum Associates, Hillsdale, NJ 1977.

[SHOR 76] Shortliffe, E. H., *Computer-Based Medical Consultations: MYCIN*, American Elsevier, New York, 1976.

[SMALL 80] Small, S., "Word Expert Parsing: A Theory of Distributed Word-Based Natural Language Understanding," TR-954, University of Maryland, 1980.

[SMITH] Smith, D. E., and Clayton, J. E., "A Frame-Based Production System Architecture," Proceedings First Annual National Conference Artificial Intelligence, Stanford University, August 1980, pp. 154-156.

[STANSFIELD 79] Stansfield, James L., "Developing Support Systems for Information Analysis," in *Artificial Intelligence, An MIT Perspective*, Winston, P. H., and Brown, R. H., (Eds.), The MIT Press, Cambridge, MA, 1979.

[STEF 79] Stefik, M., "An Examination of a Frame-Structured System," Proc. Sixth International Conference on Artificial Intelligence, Tokyo, 1979, pp. 845-852.

[STEF 81] Stefik, M., "Planning With Constraints (MOLGEN: Part 1)," *Artificial Intelligence*, Vol. 16, No. 2 (May, 1981), pp. 111-140. "Planning and Meta-Planning (MOLGEN: Part 2)," *Artificial Intelligence*, Vol. 16, No. 2 (May, 1981), pp. 141-170.

[STEINBERG 80] Steinberg, L., "Question Ordering in Mixed Initiative Program Specification Dialog," *Proceedings of the first Annual Conference on Artificial Intelligence*, Stanford, 1980.

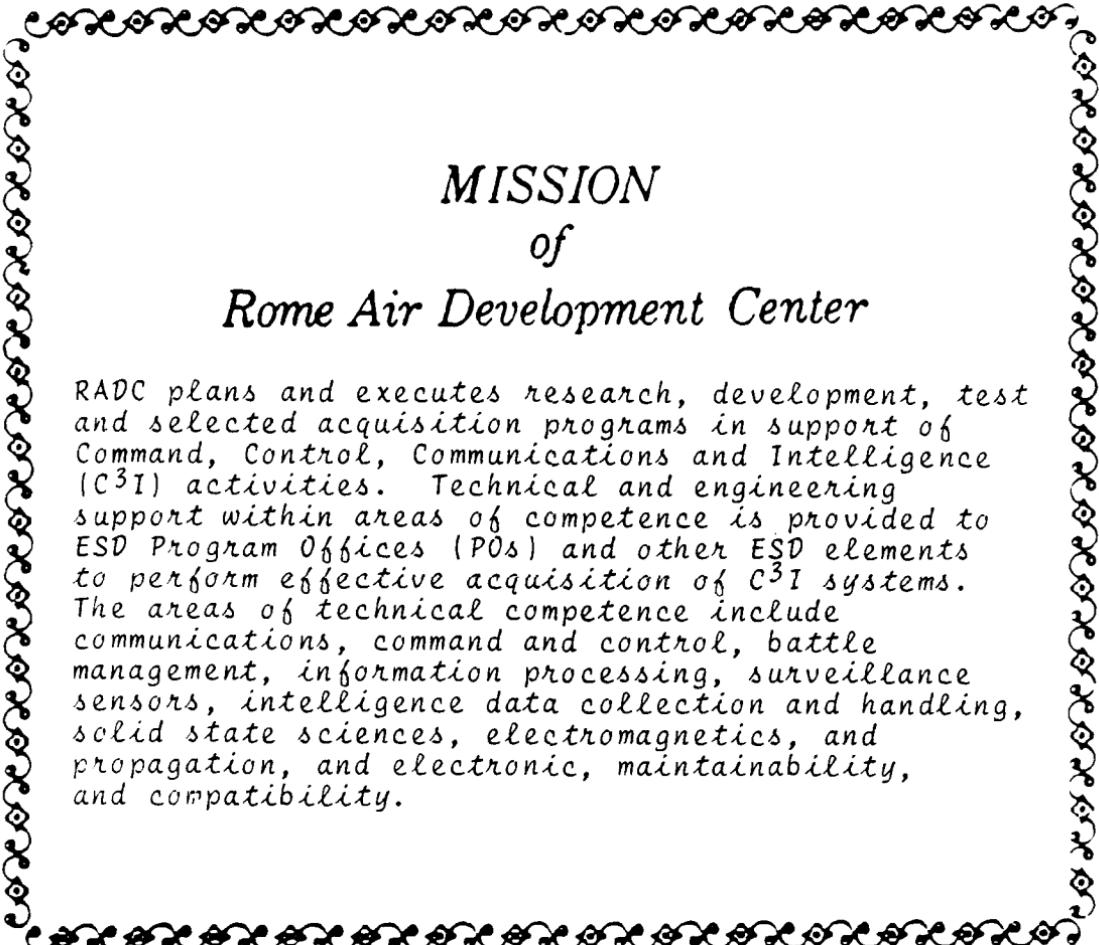
[SUSS 70] Sussman, G. J., Winograd, T., "Micro-Planner Reference Manual," MIT AI Lab Memo 203, July 1970.

[SZOL 77] Szolovits, P., Hawkinson, L. B., and Martin, W. A., "An Overview of Owl, A Language for Knowledge Representation," MIT/LCS/TM-86, MIT, Cambridge, MA, June, 1977.

[WILKS 72] Wilks, Y., "Grammar, Meaning and The Machine Analysis of Language," London, 1972.

[WOOD 70] Woods, W. A., "Transition Network Grammars for Natural Language Analysis," *Comm. ACM*, Vol. 13, No. 10, October 1970.

[WOOD 72] Woods, W. A., Kaplan, R. M. and Nash-Webber, B., "The Lunar Sciences Natural Language Information System," BBN Report 2378, Bolt, Beranek, and Newman, Inc., Cambridge, MA, 1972.



*MISSION
of
Rome Air Development Center*

RADC plans and executes research, development, test and selected acquisition programs in support of Command, Control, Communications and Intelligence (C³I) activities. Technical and engineering support within areas of competence is provided to ESD Program Offices (POs) and other ESD elements to perform effective acquisition of C³I systems. The areas of technical competence include communications, command and control, battle management, information processing, surveillance sensors, intelligence data collection and handling, solid state sciences, electromagnetics, and propagation, and electronic maintainability, and compatibility.

END

1-87

DTIC